Open Education Cup: Preface

Thank you for your interest in the Open Education Cup 2008-09 competition.

This book is designed to provide you with information about the competition as well as resources to assist you in crafting your contest submissions. This book has been divided into three chapters as follows:

Chapter 1 - About the Open Education Cup Contest
This chapter provides basic information about the Open Education Cup competition, including:

- The purpose of the competition.
- Contest rules, awards, and criteria.
- Submission instructions.
- Information on Open Education Resources (OER) and Connexions (http://cnx.org).

Chapter 2 - Book Introduction and Example Connexions Modules
This chapter includes the Parallel Computing module by Charles Koelbel. This module, which will serve as the introductory chapter in the High Performance Computing textbook comprised of winning contest entries, serves to establish the framework for contest submissions and provide a foundation upon which participants can build their contributions. This chapter also includes two exemplar modules that illustrate several Connexions features participants are encouraged to take advantage of and to serve as inspiration for crafting high-quality modules.

Chapter 3 - Authoring in Connexions
This chapter includes two brief CNXML tutorials as well as links to additional resources for Connexions authors.

This book was created in Connexions by assembling a customized combination of new and existing content modules. As you browse this booklet, you will come across several examples of the power of the Connexions platform, including:

- Automatic numbering of figures, sections, chapters, and equations.
- Automatically generated Glossary and Index sections.
- Attribution of individual modules used within the collection.
- Powerful XML technologies that allow content to be viewed in print, online or as a PDF download without any additional work by the author.
- Support for MathML content, allowing for high-fidelity representations of complex mathematical expressions.
- Multiple authoring paths, including Word and LaTeX file importers and support for contextual and/or full-source CNXML editing.

You are encouraged to view this collection online at http://cnx.org/content/col10594/latest/ to see how Connexions content can be accessed and viewed freely online, downloaded as a free PDF, or ordered as a low-cost print-on-demand book (such as the one distributed at SC08). Selected contest entries will be included in a new collection such as this one that will combine the best modules into a free, open, and fully customizable collection/textbook available to everyone globally.

We hope you will consider contributing to this project and participating in the Open Education Cup competition. If you have any questions or difficulties, please contact Jonathan Emmons at cnx@cnx.org.

Please enjoy the rest of SC08, and have a great day.

Introduction and Overview of the 2008-'09 Open Education Cup

Note: For details and the most up to date rules on how to submit a lesson (Connexions **module**) to the contest visit the Open Education Cup (http://OpenEducationCup.org) web page. The objective of this section is to provide a bird's eye view of the contest. If there is disagreement between statements made in the printed version of this document, the electronic version of this document and in the contest website, the contest website should always be considered the "gold standard."

Introduction

This booklet was created to support the 2008-'09 Open Education Cup that was launched during SC08 (http://sc08.supercomputing.org/) in Austin, Texas. The Open Education Cup was conceived of and created in an attempt to create awareness and enhance the interest in creating Open Educational Resources (OER) in the area of HPC. We hope that the effort will help the HPC community recognize and embrace the opportunity that OER represents for accelerating workforce development. OER can enable educators, instructors and learners to access a rich set of free online teaching and learning material, hence lowering the barriers to offering education and training programs.

It is critically important that we, as a community, recognize the increasing role of cyberinfrastructure in society. In particular, we must enable a larger fraction of our science and engineering graduates to become familiar with and even master the necessary technical skills to be productive citizens. The Open Education Cup will help create much needed content that can be used by teachers, instructors and learners to become at a minimum HPC literate or, better yet, capable of harnessing complex HPC resources and become expert users and programmers of current and future HPC systems. This is important, whether the objective is to harness the power of a national supercomputer with 100s of thousand of cores, exploit and program "personal supercomputers" with hardware accelerators delivering multi-

teraflop capabilities, or program personal laptops that soon will have dozens of processing cores.

The modules included in the booklet you are holding was not meant to represent specific content relevant to HPC, but was assembled in an attempt to demonstrate the power of the OER repository Connexions. This booklet, in addition to being a description of the Open Education Cup serves as a mini tutorial on OER and Connexions and includes a set of sample modules demonstrating of the capabilities of Connexions. The booklet is just a snapshot in time of what was available online. This printed collection was created using QOOP's on-demand publishing process supported by Connexions, at a fraction of the cost of a traditional textbook.

As always will be the case with open online educational resources, what you are holding in your hands may have and has likely been updated online. We urge you to visit the Open Education Cup collection at http://cnx.org/content/col10594/latest/ to review the most up-to-date content. For the ultimate definition of the rules for the contest you should always refer to http://OpenEducationCup.org.

Contest Overview

Open Educational Resources is offering the High Performance Computing community a unique opportunity to accelerate the education of the next generation science and engineering workforce. We all recognize and have read the reports reminding us that:

- Parallel computing is becoming ubiquitous. While 10 years ago only the largest servers used parallel processors, today even lonely laptops sport dual- (and soon multi-) processor chips. It is clear that soon all software, if it is to effectively use the hardware, will have to run in parallel.
- Too few students are learning parallel computing concepts. This is particularly true at the undergraduate level (and below), where few introductory textbooks or curricula tackle the subject. As long as parallel computing is considered an advanced topic, the supply of parallel software will lag behind.

- Creating teaching materials about parallel computing is difficult. Writing or updating a textbook on any subject is difficult and time-consuming at best. Keeping up with a fast-changing field like parallel computing magnifies that problem exponentially. Worse, a ubiquitous change like parallel computing forces changes in all areas of the curriculum.

It is worth noting that a number of national reports (e.g. the recent PCAST report, Leadership Under Challenge: Information Technology R&D in a Competitive World) have made similar observations.

We believe OER will be a key component for addressing these needs. The idea behind OER is to make high-quality educational material freely available to teachers and learners so that they can master and enhance the material, in the same way that the Open Source movement has furthered the production of software. Once OER content exists, it can be picked up, used, and improved by a large and growing community. This creates a vibrant "ecosystem" for teaching and learning about a topic. In the context of parallel computing, we have already mentioned the need to create educational modules. Putting parallel computing modules into an OER system will allow others to comment on them, improve them, add to them, and (most importantly) learn from them. If the parallel computing community embraces OER and starts sharing its knowledge, it can rapidly help build the education and training materials needed to train learners, disseminate new information, and take the field to new heights.

The Connexions project at Rice University is a technology platform supporting OER for authors, educators and learners. As an open source / open content educational project, Connexions can host, distribute and serve as a platform for authoring and maintaining educational material. It provides excellent support for OER, including establishing a framework for new collaborators, providing and encouraging reusable content (modules), accepting a variety of input formats and supporting in place editing. It already hosts over 7000 educational modules in areas ranging from Fourier analysis, signal processing, statistics and physics to music theory. We have chosen Connexions as the platform for the Open Education Cup in order to

have a convenient, scalable and reusable central repository for all the modules submitted.

The purpose of this competition is to jump-start an OER repository, using the Connexions platform, on High Performance Computing (HPC) and parallel computing. We think that, by offering modest inducements (monetary prizes, publications, and fame), we can quickly collect a wide variety of modules to teach parallel computing. By using Connexions tools to publish and host these Open Educational Resources, they will be freely available (published under the Creative Commons attribution license) to the students, professors, and teachers who most need them. Instructors and teachers can then choose their preferred sets of existing modules sometimes enhanced by personally authored content, thus creating a variety of courses that are engaging, comprehensive, constantly updated, and customized to their students needs. The OER repository Connexions also support translation of modules into multiple languages. While the contest entries **must be in English** we encourage authors to support language translations. In the long term, such an OER repository and the collaborative courses in it can form the foundation for a new way of teaching computer and computational science. This is admittedly a grand vision, but it is imperative that we start now to build core material for the coming wave of new computational learning.

Contest Rules

To be considered for the contest, prospective authors are required to submit entries that adhere to the following guidelines:

- Prepare/author **original** content in **English** in one of the following five (5) contest categories:
    - **Parallel Architectures** Descriptions of parallel computers and how they operate, including particular systems (e.g. multicore chips) and general design principles (e.g. building interconnection networks).
    - **Parallel Programming Models and Languages** Abstractions of parallel computers useful for programming them efficiently,

including both machine models (e.g. PRAM or LogP) and languages or extensions (e.g. OpenMP or MPI).
    - **Parallel Algorithms and Applications** Methods of solving problems on parallel computers, from basic computations (e.g. parallel FFT or sorting algorithms) to full systems (e.g. parallel databases or seismic processing).
    - **Parallel Software Tools** Tools for understanding and improving parallel programs, including parallel debuggers (e.g. TotalView) and performance analyzers (e.g. HPCtoolkit and TAU Performance System).
    - **Accelerated Computing** Hardware and associated software can add parallel performance to more conventional systems, from Graphics Processing Units (GPUs) to Field Programmable Gate Arrays (FPGAs) to Application-Specific Integrated Circuits (ASICs) to innovative special-purpose hardware.

- Derived modules (i.e., creating a new module from a previously published module) will only be considered if the module is derived from your own previously published Connexions module. Modules derived from existing Connexions modules written by other authors, while generally encouraged in an open educational repository, will not be accepted in the contest. Co-authored modules are, however, permitted.
- The content, related to one of the subject areas, **must be published** in Connexions. The author must not only upload a draft copy of the module into Connexions, but must go through the final step of publishing the module (agreeing to the Creative Commons open licensing terms adopted by Connexions) before the posted deadline. For additional information about each of the subject areas of the contest see the module Parallel Computing by Charles Koelbel.
- A single module can only be judged in one of the categories and it is the responsibility of the author to select the most appropriate category (authors must select the category in the web form discussed below). Modules sent to multiple categories will only be judged in the first category to which they are submitted.
- Authors are free to update the modules after the deadline and we encourage this. However, only the version that was current as of the

posted deadline will be judged. Connexions allows access to time-stamped versions of modules so it is possible to judge the submitted version even as updates are being made.
- For a module to be considered in the contest, the author must also fill out and submit the form entitled “Submit Module” linked from the Open Education Cup web page at http://openeducationcup.org/submitform.html. The Submit Module form must be completed after the module is published in Connexions because the URL assigned to it is not permanent until that time. Updates to a module after the first version has been published will maintain the same module number, only the version number and time stamp will be updated.
- An individual author may, and is strongly encouraged to, submit more than one module. However, the same module may only be submitted to one content category. The author may submit multiple modules to the same content categories and/or submit modules to multiple content categories. Please see the Contest Award section for award descriptions and limitations.
- Entries MUST be submitted (both the published module in Connexions and Submit Module web form) on or before 5 p.m. (US central time zone) Monday, February 2, 2009.

For a complete and final listing of all the contest rules for the Open Education Cup go to http://OpenEducationCup.org.

Contest Awards

A panel of experts in high performance computing and parallel programming will be named to serve as judges for the contest. Each module will be reviewed by at least three impartial judges. The judges select one in each of the five (5) subject categories (as defined in the Contest Rules section) as the **Best** module, resulting in five (5) Best module awards total. In addition the judges can select any number of entries for the distinction of **Honorable Mention**. Judging criteria will be posted at the Open Education Cup web page.

In each category, there will be a first prize of $500. At the judges' discretion, other modules in the category may be identified for Honorable Mention. An on-line collection, which may be printed as a book, will highlight all modules that received either recognition.

Thanks to the generous support from our sponsors (BP, Chevron, Connexions, NVIDIA, Rice, Sun Microsystems, Total S.A. and WesternGeco), we will be awarding prizes to contest winners according to the following guidelines:

- Each of the five (5) modules selected as Best will receive a prize in the amount of US $500.
- A contestant may only receive **one** prize in the category **Best**.
- A contestant may be nominated for and receive more than one **Honorable Mention**.
- A module with multiple authors must designate one author as the lead author. If a module with multiple authors is selected to receives one of the prizes the organizer will issue the prize to the lead author. It is the **sole responsibility** of the lead author in collaboration with the co-authors to agree on how to divide the prize.
- A Connexions collection (book) containing all the modules given the distinction Best and Honorable Mention will be created and made available in the Connexions repository after the end of the contest.
- All awards will be publicized on the Open Education Cup contest website, through the published collection referenced above, open education conferences, appropriate HPC conferences focused on education and content creation and other promotional materials.

Judging Criteria

A panel of distinguished parallel computing experts from academia and industry will read and judge all modules. Modules within a category will only be compared against other modules in the same category (e.g. Parallel Architecture modules will **not** be compared to Parallel Software Tools modules). The primary judging criteria will be:

- **Appropriateness:** The module should be relevant to the category in which it is entered, as well as to parallel computing generally. Its topic should be considered of some importance in the field, though it need not be “the” central concept.
- **Correctness:** The module should be technically accurate and cite external references where appropriate. Where there is controversy in a field, it may take a particular stand. However, it should note where there are significant differences with other sources.
- **Clarity:** The module should be easy for a learner to understand. Where appropriate, it should note any prerequisite knowledge. A module may target any audience from elementary school to graduate student.
- **Presentation:** The module should make good use of formatting and auxiliary files. It need not use every feature of Connexions (e.g. bibliographies, graphics support, equations), but where a feature is used it should be attractive (e.g. clearly legible figures). Creative use of multimedia enhancements of the material is encouraged.

What is a module?

There is no hard and fast definition of the length or size of a module; it could be half a page of text or it could be the equivalent of 10 or more pages of text. Somewhere in the middle is most likely -- 3-6 pages. You can think of the proper length as that which is needed to treat a concept in sufficient detail so someone can read it and make sense of it. If the module depends on the understanding of other concepts, these should generally not be treated in your module but you can assume will be treated in a separate module. For the purpose of this contest you can assume that pre-requisites exist and the judges will not be judging you on whether or not the pre-requisites exist at the time you write your module. Over time these pre-requisite modules will be written, either by you or others.

Submitting Content

Open Education Cup participants are strongly encouraged to publish their modules well ahead of the submission deadline. Authors new to

Connexions may need time to learn and adjust to the authoring interface as well as the features and options provided by the CNXML language, and those wishing to upload Word, OpenOffice Writer, or LaTeX documents will also need time to prepare those files using the importer templates provided as well as perform post-import edits as necessary. By starting the authoring and publication process early, contest participants will ensure that they have the time to work through any difficulties and the opportunity to contact Connexions for additional support as necessary.

Authors are strongly encouraged to include relevant information such as bibliography, glossary, links to prerequisite material, and related links. If you have any questions regarding the authoring process, or if you experience any trouble publishing your contest entries, please do not hesitate to contact Jonathan Emmons, Connexions' community development specialist, at cnx@cnx.org for assistance. Connexions will be offering a series of web-training sessions for those interested, and will be available for individual support as necessary. Again, participants are strongly encouraged to begin the authoring and publication process early in order to identify barriers and request assistance in a timely manner.

To enter a module in the contest, authors MUST complete BOTH of the following steps (in the order specified):

1. Publish the module in Connexions
2. Fill out and submit the "Submit Module" form at http://openeducationcup.org/submitform.html

Modules must be published once before they can be submitted to the contest (though authors may continue to update their Connexions modules after entering the contest). Only URLs for published modules (of the form http://cnx.org/content/[moduleID]/latest/) will be accepted; URLs to modules in workgroups or workspaces are not publicly viewable and will not be considered valid entries.

Getting Started

For a quick start on creating a Connexions module visit the Create a Module in Minutes tutorial at http://cnx.org/help/ModuleInMinutes or visit the New Author Guide at http://cnx.org/content/col10404/latest/ for a short course to get stated using Connexions. For a more detailed description of Connexions take a look at the Connexions Tutorial and Reference material at http://cnx.org/content/col10151/latest/. We also strongly encourage you to review the module on Importing and Exporting at http://cnx.org/help/ImportAndExport to get up to speed on the most effective way of authoring new content or leveraging content that you may already have written in Word or LaTeX and want to make available in Connexions.

A number of modules have been include as a resource that you may find useful as you prepare your contest entry, including:

- The chapter Parallel Computing by Charles Koelbel provides a solid framework for the envisioned collection of modules.
- Resources, help pages, and tutorials for Connexions authors.
- Exemplar modules that illustrate how to leverage Connexions to produce high-quality educational materials.

Even though most of the example modules we have chosen to include in this book do not relate directly to the topic of high performance computing and parallel computing, we hope that the modules included in this collection will help stimulate ideas for what might be possible as well as give you the opportunity to see how modules are rendered both in print as well as online. We strongly encourage authors to maximally leverage the repositories ability to support dynamic multimedia content. Research shows that the education and learning experience can be significantly enhanced if the learner has access to material that stimulates more than one learning style (e.g., visual, auditory, reading/writing-preference or kinesthetic or tactile).

Sponsor Recognition

Finally we want to thank all the sponsors for their generous support. Without the support of BP, Chevron, Connexions, NVIDIA, Rice, Sun

Microsystems, Total S.A., and WesternGeco we would not have been able to kick-start the Open Education Cup at SC08 in Austin, Texas. Without the interest of our sponsors and that of the community for advancing our ability to educate and train the next generation employees we could not have brought you this opportunity. Please make sure you thank each of our sponsors when you meet them at SC08.

Happy authoring!

Open Education Cup: What is OER?

What are Open Education Resources (OER)?

The William and Flora Hewlett Foundation provides the following definition for OER:

"OER are teaching, learning and research resources that reside in the public domain or have been released under an intellectual property license that permits their free use or re-purposing by others. Open educational resources include full courses, course materials, modules, textbooks, streaming videos, tests, software, and any other tools, materials or techniques used to support access to knowledge."

We believe that opening educational materials will not only reduce the cost of education for schools, districts, and students by providing free access to textbooks and other commonly used resources, but also help to improve the quality and relevance of these materials by encouraging contributions from all community members. Rather than limiting ideas those approved by a small group of publishers, OER allows educational communities to pool individual resources from around the world to bring together the best ideas.

This model encourages author participation by making it easier to publish and update content, allowing them to share their ideas with the world without navigating the many barriers associated with traditional publication models. Educators can use OER as a way of accessing the most up-to-date, relevant information available, and customizing that content for the specific needs of their students. Students can take advantage of the zero-dollar price tag to gain access to educational resources regardless of income, making a high-quality education more affordable and available for everyone.

Why Connexions?

The Open Education Cup competition is using Connexions as the framework for submitting and publishing all contest entries. Modules submitted to Connexions are made available to the public under the Creative Commons attribution license, allowing students, educators, and

researchers to read, extend, or repurpose the content for any purpose provided they maintain proper attribution.

There are several platforms that provide access to OER content; however, there are a number of specific factors that make Connexions an ideal choice for this type of competition, specifically:

License
Many sites place restrictions on the way content can be used. By releasing all content under the Creative Commons attribution (CC-by) license, users are able to use and build upon content in any way they see fit, providing much-needed flexibility when crafting personalized learning materials.

Structure
Connexions content is stored as a series of modules that can be used independently or as part of a larger collection (such as a textbook, journal, or online course). By breaking down ideas into smaller pieces, users are able to mix and match works from other authors to create customized learning materials using the best ideas available from the content commons.

Delivery
Users can access Connexions content in one of three ways: as a free online e-book, a free downloadable PDF, or as a low-cost print-on-demand publication orderable from the collection home page. Content can also be freely copied and made available through other channels, such as organizational websites or other OER distributers, provided that proper attribution is maintained.

- For more information about the Connexions project, please visit http://cnx.org/aboutus/.
- For questions help using Connexions, please visit http://cnx.org/help/, or contact Jonathan Emmons at jonathan.emmons@cnx.org.

Other Approaches to OER

The following modules provide some additional information regarding OER you may find interesting or useful:

- OER Introduction (by Judy Baker)
- What are Open Textbooks? (by ISKME)

Parallel Computing

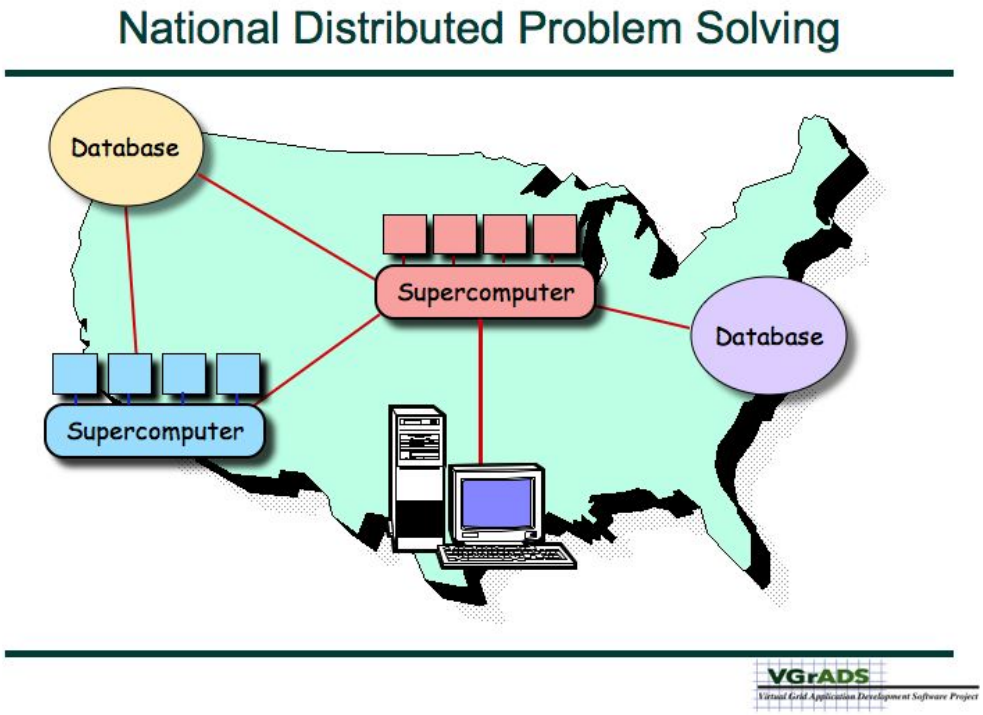
What is parallel computing, and why should I care?

Parallel computing simply means using more than one computer processor to solve a problem. Classically, computers have been presented to newcomers as "sequential" systems, where the processor does one step at a time. There are still machines and applications where this is true, but today most systems have parallel features. Some examples are shown below. [footnote]

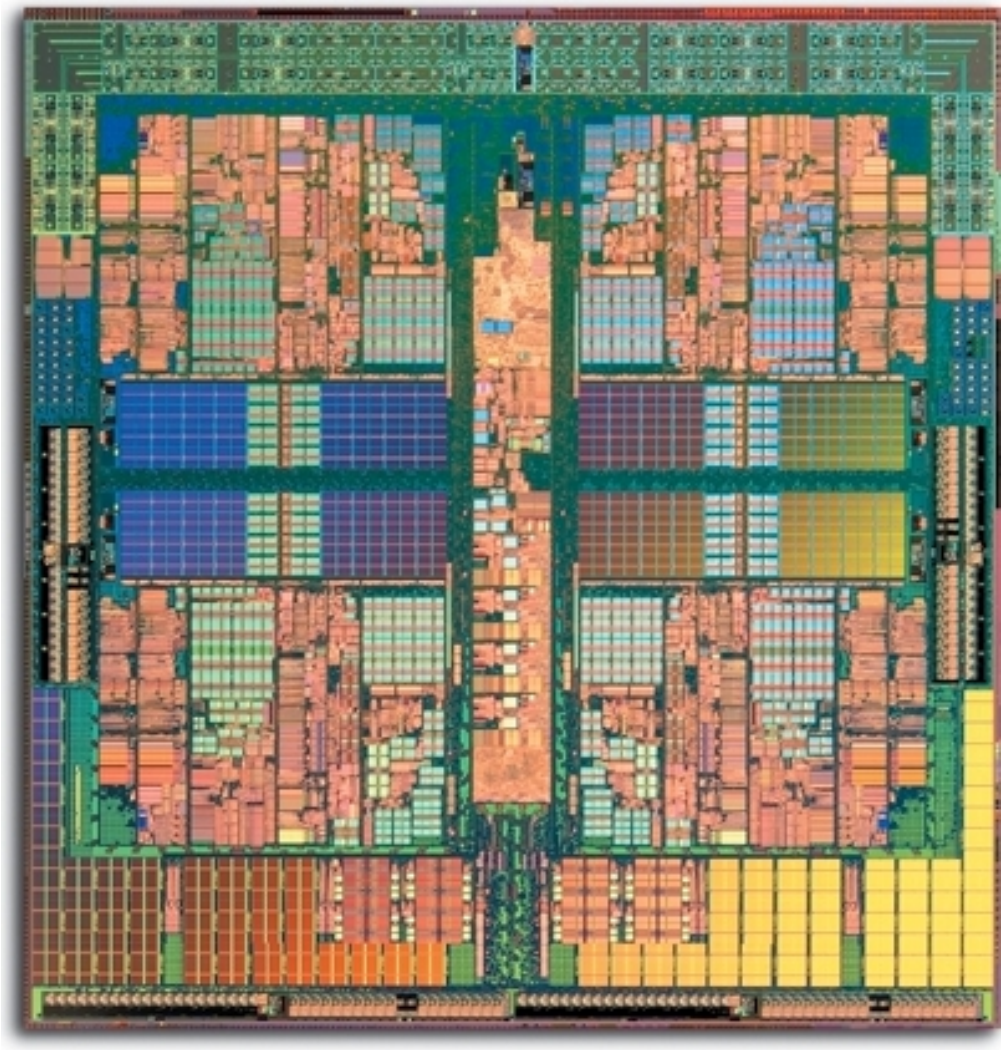
We hasten to point out that, although we use certain commercial systems and chips as examples, they are by no means the only ones. Additional examples in all categories are available from other companies. The Open Education Cup would welcome modules on any type of parallel system.

Supercomputers achieve astounding speed on scientific computations by using amazing numbers of processors. The Roadrunner system at Los Alamos National Laboratory ([link]) is currently the world's fastest computer, with over 100,000 processor cores combining to reach 1 PetaFLOPS (10^{15} floating point operations per second).

Servers use multiple processors to handle many simultaneous requests in a timely manner. For example, a web server might use dozens of processors to supply hundreds of pages per second. A 64-processor server ([link]) could supply an average university department’s need for computation, file systems, and other support.



Grid computing combines distributed computers (often from different organizations) into one unified system. One vision ([link]) is that a scientist working at her desk in Houston can send a problem to be solved by supercomputers in Illinois and California, accessing data in other locations. Other grid computing projects, such as the World Community Grid, seek to use millions of idle PCs to solve important problems, such as searching for candidate drug designs.



Multicore chips include several processor cores on a single computer chip. Even a laptop (or other small system) that uses a multicore chip is a parallel computer. For example, the Quad-core AMD Opteron processor ([link]) is one of a family of chips with 4 cores. As another example, the IBM Cell processor chips used in the Roadrunner supercomputer have 8 “Synergistic Processing Elements” (i.e. fast cores) each, plus 1 Power Processing Element (which controls the others).

Graphics processors (often called GPUs) are a special type of chip originally designed for image processing. They feature massive numbers of scalar processors to allow rendering many pixels in parallel. For example, The NVIDIA Tesla series ([link]) boasts 240 processors. Such special-purpose parallel chips are the basis for accelerated computing.

In effect, all modern computing is parallel computing.

Getting the most out of a parallel computer requires finding and exploiting opportunities to do several things at once. Some examples of such opportunities in realistic computer applications might include:

- Data parallelism – updating many elements of a data structure simultaneously, like a image processing program that smoothes out pixels
- Pipelining – using separate processors on different stages of a computation, allowing several problems to be “in the pipeline” at once
- Task parallelism – assigning different processors to different conceptual tasks, such as a climate simulation that separates the atmospheric and oceanic models
- Task farms – running many similar copies of a task for later combining, as in Monte Carlo simulation or testing design alternatives
- Speculative parallelism – trying alternatives that may not be necessary to pre-compute possible answers, as in searching for the best move in a game tree

However, doing this may conflict with traditional sequential thinking. For example, a speculative program may do more total computation than its sequential counterpart, albeit more quickly. As another example, task-

parallel programs may repeat calculations to avoid synchronizing and moving data. Moreover, some algorithms (such as depth-first search) are theoretically impossible to execute in parallel, because each step requires information from the one before it. These must be replaced with others (such as breadth-first search) that can run in parallel. Other algorithms require unintuitive changes to execute in parallel. For example, consider the following program:

```
for i = 2 to N
  a[i] = a[i] + a[i-1]
end
```

One might think that this can only be computed sequentially. After all, every element `a[i]` depends on the one computed before it. But this is not so. We leave it to other module-writers to explain how the elements `a[i]` can all be computed in $\log(N)$ data-parallel steps; for now, we simply note that the parallel program is not a simple loop. In summary, parallel computing requires parallel thinking, and this is very different from the sequential thinking taught to our computer and computational scientists.

This brings us to the need for new, different, and hopefully better training in parallel computing. We must train new students to embrace parallel computing if the field is going to advance. Equally, we must retrain programmers working today if we want new programs to be better than today. To that end, the Open Education Cup is soliciting educational modules in five areas at the core of parallel computing:

- Parallel Architectures: Descriptions of parallel computers and how they operate, including particular systems (e.g. multicore chips) and general design principles (e.g. building interconnection networks). More information about this area is in the Parallel Architectures section.
- Parallel Programming Models and Languages: Abstractions of parallel computers useful for programming them efficiently, including both machine models (e.g. PRAM or LogP) and languages or extensions

(e.g. OpenMP or MPI). More information about this area is in the Parallel Programming Models and Languages section.
- Parallel Algorithms and Applications: Methods of solving problems on parallel computers, from basic computations (e.g. parallel FFT or sorting algorithms) to full systems (e.g. parallel databases or seismic processing). More information about this area is in the Parallel Algorithms and Applications section.
- Parallel Software Tools: Tools for understanding and improving parallel programs, including parallel debuggers (e.g. TotalView) and performance analyzers (e.g. HPCtoolkit and TAU Performance System). More information about this area is in the Parallel Software Tools section.
- Accelerated Computing: Hardware and associated software can add parallel performance to more conventional systems, from Graphics Processing Units (GPUs) to Field Programmable Gate Arrays (FPGAs) to Application-Specific Integrated Circuits (ASICs) to innovative special-purpose hardware. More information about this area is in the Accelerated Computing section.

Entries that bridge the gap between two areas are welcome. For example, a module about a particular accelerated computing system might describe its accompanying language, and thus fall into both the Parallel Programming Models and Languages and Accelerated Computing categories. However, such modules can only be entered in one category each. To be eligible for awards in multiple categories, the module would need to be divided into appropriate parts, each entered in one category.

Parallel Architectures

Today, nearly all computers have some parallel aspects. However, there are a variety of ways that processors can be organized into effective parallel systems. The classic classification of **parallel architectures** is Flynn's taxonomy based on the number of distinct instruction and data streams supplied to the parallel processors.

- Single Instruction, Single Data (SISD) – These are sequential computers. This is the only class that is not a parallel computer. We

include it only for completeness.

- Multiple Instruction, Single Data (MISD) – Several independent processors work on the same stream. Few computers of this type exist. Arguably the clearest examples are fault tolerant systems that replicate a computation, comparing answers to detect errors; the flight controller on the US Space Shuttle is based on such a design. Pipelined computations are also sometimes considered MISD. However, the data passed between stages of the pipeline has been changed, so the "single" data aspect is murky at best.
- **Single Instruction, Multiple Data (SIMD)** – A central controller sends the same stream of instructions to a set of identical processors, each of which operates on its own data. Additional control instructions move data or exclude unneeded processors. At a low level of modern architectures, this is often used to update arrays or large data structures. For example, GPUs typically get most of their speed from SIMD operation. Perhaps the most famous large-scale SIMD computer was the Thinking Machines CM-2 in the 1980s, which boasted up to 65,536 bit-serial processors.
- **Multiple Instruction, Multiple Data (MIMD)** – All processors execute their own instruction stream, each operating on its own data. Additional instructions are needed to synchronize and communicate between processors. Most computers sold as "parallel computers" today fall into this class. Examples include the supercomputers, servers, grid computers, and multicore chips described above.

Hierarchies are also possible. For example, a MIMD supercomputer might include SIMD chips as accelerators on each of its boards.

Beyond general class, many architectural decisions are critical in designing a parallel computer architecture. Two of the most important include:

- Memory hierarchy and organization. To reduce data access time, most modern computers use a hierarchy of caches to keep frequently-used data accessible. This becomes particularly important in parallel computers, where many processors mean even more accesses. Moreover, parallel computers must arrange for data to be shared between processors. **Shared memory architectures** do this by

allowing multiple processors to access the same memory. **Nonshared memory architectures** allot each processor its own memory, and require explicit communication to move data to another processor. A hybrid approach – non-uniform shared memory – places memory with each processor for fast access, but allows slower access to other processors' memory.

- Interconnection topology. To communicate and synchronize, processors in a parallel computer need a connection with each other. However, as a practical matter not all of these can be direct connections. Thus is born the need for interconnection networks. For example, interconnects in use today include simple buses, crossbar switches, token rings, fat trees, and 2- and 3-D torus topologies. At the same time, the underlying technology or system environment may affect the networks that are feasible. For example, grid computing systems typically have to accept the wide-area network that they have available.

All of these considerations (and more) can be formalized, quantified, and studied.

The Open Education Cup will accept entries relevant to any of the above architectures (except SISD). This might include case studies of parallel computers, comparisons of architectural approaches, design studies for future systems, or parallel computing hardware issues that we do not touch on here.

Parallel Programming Models and Languages

Parallel computers require software in order to produce useful results. Writing that software requires a means of expressing it (that is, a language), which must in turn be based on an idea of how it will work (that is, a model). While it is possible to write programs specific to a given parallel architecture, many would prefer a more general model and language, just as most sequential programmers use a general language rather than working directly with assembly code.

Unfortunately, there is no single widely-accepted model of parallel computation comparable to the sequential Random Access Machine (RAM) model. In part, this reflects the diversity of parallel computer architectures and their resulting range of operations and costs. The following list gives just a few examples of the models that have been suggested.

- Communicating Sequential Processes (CSP). This is a theoretical model of concurrent (that is, parallel) processes interacting solely through explicit messages. In that way, it is a close model of **nonshared memory architectures**. As the CSP model has developed, however, its use has emphasized validation of system correctness rather than development of algorithms and programs.
- Parallel Random Access Machine (PRAM). In this model, all processors operate asynchronously and have constant-time access to shared memory. This is similar to a **shared memory architecture**, and is therefore useful in describing parallel algorithms for such machines. However, PRAM abstracts actual shared memory architectures by assuming that all memory can be accessed at equal cost, which is generally not true in practice.
- Bulk Synchronous Processes (BSP). In this model, all processors operate asynchronously and have direct access to only their own memory. Algorithms proceed in a series of "supersteps" consisting of local computation, communication exchanges, and barrier synchronizations (where processors wait at the barrier for all others to arrive). This can model either shared or non-shared memory **MIMD architectures**, but simplifies many issues in organizing the communication and synchronization.
- LogP (for Latency, overhead, gap, Processors). This model can be thought of as a refinement of BSP. LogP allows a more detailed treatment of architectural constraints such as interconnect network capacity. It also allows more detailed scheduling of communication, including overlap of computation with communication. LogP can model shared- and non-shared memory MIMD machines, but abstracts the specific topology of the network. (The capitalization is a pun on $O(\log P)$ theoretical bounds).

Programming languages and systems for parallel computers are equally diverse, as the following list shows.

- Libraries can encapsulate parallel operations. Libraries of synchronization and communication primitives are especially useful. For example, the MPI library provides the send and receive operations needed for a CSP-like message passing model. When such libraries are called from sequential languages, there must be a convention for starting parallel operations. The most common way to do this is the Single Program Multiple Data (SPMD) paradigm, in which all processors execute the same program text.
- Extensions to sequential languages can follow a particular model of parallel computation. For example, OpenMP reflects a PRAM-like shared memory model. High Performance Fortran (HPF) (see also) was based on a data-parallel model popularized by CM Fortran. Common extensions include parallel loops (often called "`forall`" to evoke "`for`" loops) and synchronization operations.
- Entire new languages can incorporate parallel execution at a deep level, in forms not directly related to the programming models mentioned above. For example, Cilk is a functional language that expresses parallel operations by independent function calls. Sisal was based on the concept of streams, in the elements in a series (stream) of data could be processed independently. Both languages have been successfully implemented on a variety of platforms, showing the value of a non-hardware-specific abstraction.
- Other languages more directly reflect a parallel architecture, or a class of such architectures. For example, Partitioned Global Address Space (PGAS) languages like Co-Array Fortran, Chapel, Fortress, and X10 consider memory to be shared, but each processor can access its own memory much faster than other processors' memory. This is similar to many non-uniform shared memory architectures in use today. CUDA is a rather different parallel language designed for programming on GPUs. It features explicit partitioning of the computation between the host (i.e. the controller) and the "device" (i.e. GPU processors). Although these languages are to some extent hardware-based, they are general enough that implementations on other platforms are possible.

Neither of the lists above is in any way exhaustive. The Open Education Cup welcomes entries about any aspect of expressing parallel computation. This includes descriptions of parallel models, translations between models, descriptions of parallel languages or programming systems, implementations of the languages, and evaluations of models or systems.

Parallel Algorithms and Applications

Unlike performance improvements due to increased clock speed or better compilers, running faster on parallel architectures doesn't just happen. Instead, **parallel algorithms** have to be devised to take advantage of multiple processors, and applications have to be updated to use those algorithms. The methods (and difficulty) of doing this vary widely. A few examples show the range of possibilities.

Some theoretical work has taken a data-parallel-like approach to designing algorithms, allowing the number of processors to increase with the size of the data. For example, consider summing the elements of an array. Sequentially, we would write this as

```
x = 0
for i = 1 to n
  x = x + a[i]
end
```

Sequentially, this would run in $O(n)$ time. To run this in parallel, consider `n` processes, numbered 1 to `n`, each containing original element `a[i]`. We might then perform the computation as a data parallel tree sum, with each node at the same level of the tree operating in parallel.[footnote]
We use "forall" to denote parallel execution in all examples, although there are many other constructs.

```
step = 1
forall i = 1 to n do
```

```
  xtmp[i] = a[i]
end
while step < n do
  forall i = 1 to n-step by 2*step do
    xtmp[i] = xtmp[i] + xtmp[i+step]
  end
  step = step * 2
end
x = xtmp[1]
```

This takes $O(\log n)$ time, which is optimal in parallel. Many such algorithms and bounds are known, and classes (analogous to P and NP) can be constructed describing "solvable" parallel problems.

Other algorithmic work has concentrated on mapping algorithms to more limited set of parallel processors. For example, the above algorithm might be mapped onto p processors in the following way.

```
forall j = 1 to p do
  lo[j] = 1+(j-1)*n/p
  hi[j] = j*n/p
  xtmp[j] = 0
  for I = lo[j] to hi[j] do
    xtmp[j] = xtmp[j] + a[i]
  end
  do
    if j=1
    then step = 1
    barrier_wait()
    while step[j] < n do
      if j+step[j]<p and j mod (step[j]*2) = 1
      then xtmp[j] = xtmp[j] + xtmp[j+step[j]]
    end
    step[j] = step[j] * 2
    barrier_wait()
```

```
  end
end
x = xtmp[1]
```

Note that the barrier synchronizations are necessary to ensure that no processor `j` runs ahead of the others, thus causing some updates of `xtmp[j]` to use the wrong data values. The time for this algorithm is $O(n/p + \log p)$. This is optimal for operation count, but not necessarily for number of synchronizations.

Many other aspects of parallel algorithms deserve study. For example, the design of algorithms for other important problems is a vast subject, as is describing general families of parallel algorithms. Analyzing algorithms with respect to time, memory, parallel overhead, locality, and many other measures is important in practice. Practical implementation of algorithms, and measuring those implementations, is particularly useful in the real world. Describing the structure of a full parallel application provides an excellent guidepost for future developers. The Open Education Cup invites entries on all these aspects of parallel algorithms and applications, as well as other relevant topics.

Parallel Software Tools

Creating a **parallel application** is complex, but developing a correct and efficient parallel program is even harder. We mention just a few of the difficulties in the following list.

- All the bugs that can occur in sequential programming – such as logic errors, dangling pointers, and memory leaks – can also occur in parallel programming. Their effects may be magnified, however. As one architect of a grid computing system put it, “Now, when we talk about global memory, we really mean **global**.”
- Missing or misplaced synchronization operations are possible on all **MIMD** architectures. Often, such errors lead to deadlock, the condition where a set of processors are forever waiting for each other.
- Improper use of synchronization can allow one processor to read data before another has initialized it. Similarly, poor synchronization can

allow two processors to update the same data in the wrong order. Such situations are called "race conditions", since the processors are racing to access the data.

- Even with proper synchronization, poor scheduling can prevent a given processor from ever accessing a resource that it needs. Such a situation is called starvation.
- Correct programs may still perform poorly because one processor has much more work than others. The time for the overall application then depends on the slowest processor. This condition is called load imbalance.
- Processors may demand more out of a shared resource, such as a memory bank or the interconnection network, than it can provide. This forces some requests, and thus the processors that issued them, to be delayed. This situation is called contention.
- Perhaps worst of all, the timing between MIMD processors may vary from execution to execution due to outside events. For example, on a shared machine there might be additional load on the interconnection network from other users. When this happens, any of the effects above may change from run to run of the program. Some executions may complete with no errors, while others produce incorrect results or never complete. Such “Heisenbugs” (named for Werner Heisenberg, the discoverer of the uncertainty principle in quantum mechanics) are notoriously difficult to find and fix.

Software tools like debuggers and profilers have proved their worth in helping write sequential programs. In parallel computing, tools are at least as important. Unfortunately, parallel software tools are newer and therefore less polished than their sequential versions. They also must solve some additional problems, beyond simply dealing with the new issues noted above.

- Parallel tools must handle multiple processors. For example, where a sequential debugger sets a checkpoint, a parallel debugger may need to set that checkpoint on many processors.
- Parallel tools must handle large data. For example, where a sequential trace facility may produce megabytes of data, the parallel trace may

produce megabytes for each of hundreds of processors, adding up to gigabytes.
- Parallel tools must be scalable. Just as some bugs do not appear in sequential programs until they are run with massive data sets, some problems in parallel programs do not appear until they execute on thousands of processors.
- Parallel tools must avoid information overload. For example, where a sequential debugger may only need to display a single line number, its parallel counterpart may need to show line numbers on dozens of processors.
- Parallel tools must deal with timing and uncertainty. While it is rare for a sequential program's behavior to depend on time, this is the common case for parallel programs.

Current parallel tools - for example Cray Apprentice2, HPCToolkit, TAU, TotalView, and VTune - have solved some of these problems. For example, data visualization has been successful in avoiding information overload and understanding large data sets. However, other issues remain difficult research problems.

The Open Education Cup welcomes entries related to the theory and practice of parallel software tools. This includes descriptions of existing debuggers, profilers, and other tools; analysis and visualization techniques for parallel programs and data; experiences with parallel tools; and any other topic of interest to tool designers or users.

Accelerated Computing

Accelerated computing is a form of **parallel computing** that is rapidly gaining popularity. For many years, some general-purpose computer systems have included accelerator chips to speed up specialized tasks. For example, early microprocessors did not implement floating-point operations directly, so machines in the technical market often included floating-point accelerator chips. Today, microprocessors are much more capable, but some accelerators are still useful. The list below suggests a few of them.

- Graphics Processing Units (GPUs) provide primitives commonly used in graphics rendering. Among the operations sped up by these chips are texture mapping, geometric transformations, and shading. The key to accelerating all of these operations is parallel computing, often realized by computing all pixels of a display or all objects in a list independently.
- Field Programmable Gate Arrays (FPGAs) provide many logic blocks linked by an interconnection fabric. The interconnections can be reconfigured dynamically, thus allowing the hardware datapaths to be optimized for a given application or algorithm. When the logic blocks are full processors, the FPGA can be used as a parallel computer.
- Application Specific Integrated Circuits (ASICs) are chip designs optimized for a special use. ASIC designs can now incorporate several full processors, memory, and other large components. This allows a single ASIC to be a parallel system for running a particular algorithm (or family of related algorithms).
- Digital Signal Processor (DSP) chips are microprocessors specifically designed for signal processing applications such as sensor systems. Many of these applications are very sensitive to latency, so performace of computation and data transfer is heavily optimized. DSPs are able to do this by incorporating **SIMD parallelism** at the instruction level and pipelining of arithmetic units.
- Cell Broadband Engine Architecture (CBEA, or simply Cell) is a relatively new architecture containing a general-purpose computer and several streamlined coprocessors on a single chip. By exploiting the **MIMD parallelism** of the coprocessors and overlapping memory operations with computations, the Cell can achieve impressive performance on many codes. This makes it an attractive adjunct to even very capable systems.

As the list above shows, accelerators are now themselves parallel systems. They can also be seen as a new level in hierarchical machines, where they operate in parallel with the host processors. A few examples illustrate the possibilities.

- General Purpose computing on GPUs (usually abbreviated as GPGPU) harnesses the computational power of GPUs to perform non-graphics

calculations. Because many GPU operations are based on matrix and vector operations, this is a particularly good match with linear algebra-based algorithms.
- Reconfigurable Computing uses FPGAs adapt high-speed hardware operations for the needs of an application. As the application goes through phases, the FPGA can be reconfigured to accelerate each phase in turn.
- The Roadrunner supercomputer gets most of its record-setting performance from Cell processors used as accelerators on its processing boards.

The Open Education Cup welcomes entries describing any aspect of accelerated computing in parallel systems. We are particularly interested in descriptions of systems with parallel accelerator components, experience with programming and running these systems, and software designs that automatically exploit parallel accelerators.

Glossary

MIMD
: Multiple Instruction Multiple Data; a type of parallel computer in which the processors run independently, possibly performing completely different operations, each on its own data.

nonshared memory
: a type of parallel computer in which each processor can directly access only its own section of memory; data to be shared with other processors must be explicitly copied to other memory areas.

parallel
: Performing more than one operation at a time; (of computers) having more than one processing unit.

Example:

> "The parallel computer had 1024 CPU chips, each with 2 processor cores, allowing 2048 simultaneous additions."

shared memory
: a type of parallel computer in which all processors can access a common area of memory.

SIMD
: Single Instruction Multiple Data; a type of parallel computer in which all processors execute the same instruction simultaneously, each on its own data.

Open Education Cup: Module Exemplars

The following pages provide a few examples of content that is available in the Connexions repository. These modules were chosen in order to illustrate specific features of the site and the ways authors are able to leverage CNXML in order to create powerful yet flexible educational content. We encourage you to review these modules and use them as a starting point for ideas as you prepare your contest entries.

Note: Due to space limitations, not all of the exemplars listed below have been included in this collection. You can view all of these modules online using the URLs provided.

Selected exemplars include:

1. Task Parallelism (by Jeff Meisel)
An effective use of a straightforward text and image presentation style. View this module online at http://cnx.org/content/m15580/latest/.

2. Analysis of Shared Memory Multiprocessors (by Bart Sinclair)
A discussion taking advantage of CNXML lists, examples, tables, and MathML support. View this module online at http://cnx.org/content/m10846/latest/.

3. Two Basic Rules of Probability (by Barbara Illowsky and Susan Dean)
A module featuring the use of CNXML sections, examples, exercises, and glossary terms. View this module online at http://cnx.org/content/m16847/latest/.

4. Half Steps and Whole Steps (by Catherine Schmidt-Jones)
An example of non-technical content in Connexions making use of images, audio media, and examples. View this module online at http://cnx.org/content/m10866/latest/.

5. Fourier Analysis in Complex Spaces (by Michael Haag and Justin Romberg)
An example making effective use of sections, subsections, examples, notes, rules (theorems), and subfigures. View this module online at http://cnx.org/content/m10784/latest/.

6. Análisis de Fourier en Espacios Complejos
A derivation of the previous module (translated by Fara Meza and Erika Jackson) illustrating how derived copies of modules can be used by other Connexions users to repurpose content, reach new audiences, and facilitate an open exchange of ideas. View this module online at http://cnx.org/content/m12848/latest/.

7. Elementary Algebra: Solving Linear Equations in One Variable
An example of a module taking full advantage of several CNXML features. This module features multimedia elements as well as advanced MathML display features. View this module online at http://cnx.org/content/m18033/latest/.

Analysis of Shared Memory Multiprocessors

A shared-memory multiprocessor consists of a number of processors accessing one or more shared memory modules. The processors can be physically connected to the memory modules in a variety of ways, but logically every processor is connected to every module.

We can use any of several metrics to evaluate the performance of a multiprocessor system. In order to keep the model and its implementation manageable, we will focus on the degree of parallelism among the memory modules; that is, we are interested in determining the average number of memory modules that are being accessed simultaneously. The model is as follows:

1. The system has p processors and m memory modules. Each processor can send requests for access to any memory module.
2. Memory module accesses are synchronized; two modules servicing access requests at the same time start and complete their accesses together. Memory accesses always take one time unit (one memory cycle).
3. Processors are infinitely fast. When a processor's access request has been serviced by a memory module, the processor immediately generates a new request.
4. Processors send requests with equal probability to each module. The module chosen for a new request is independent of the module chosen for any other request.
5. A processor may have only one outstanding request at a time.
6. A memory module may service only one request at a time. If more than one request is queued at a memory module at the beginning of a memory cycle, the module selects a request to service at random. All requests not serviced during the cycle remain queued at the module at the beginning of the next cycle.

Let B be the average number of memory modules busy servicing an access request during a memory cycle. Regardless of the number of processors and memory modules, we can implement the model as a discrete-time Markov chain in which the information contained in a state includes the number of memory modules that have requests waiting at the beginning of a memory cycle. Hence, a state also describes the number of modules that will be busy during the next memory cycle. We can find B by solving for the steady state solution of the Markov chain and computing

$$B = \sum_{\text{all states i}} \Pr[\text{state i}]\text{number of busy modules in state i}$$

We know that the Markov chain has a steady state solution (is ergodic) since the chain will be finite and at least one state will have a self-loop. The chain is finite because the number of ways that processors' access requests can be distributed among the modules is finite. Any state that represents at least one request at each memory module will have a self-loop, since any processor that has its request satisfied during a cycle beginning in that state has a non-zero probability of sending its next access request to the same module.

As usual, we proceed in three steps. We first determine, for a given p and m, the set of all states in the Markov chain. Then, we compute the probabilities for all allowed single step state transitions. Finally, we solve the Markov chain for the steady state probabilities.

It is not possible in general to implement the model using states that only tells us how many modules are going to be busy during the next cycle. Consider, for example, a model for a system with 4

processors and 2 memory modules. A state that only tells us that two modules will be busy during the next cycle does not include enough information to tell us the probabilities of being in the various states at the beginning of the next cycle. This is because two modules will be busy during a cycle that starts with requests distributed among the modules in either of two ways:

1. three requests at one module, one request at the other module
2. two requests at one module, two requests at the other module

If we start a cycle with the four requests distributed as in (1), at the end of the cycle we have two requests left at one module, no requests at the other module, and two processors ready to make new requests. The next cycle may start with the four requests distributed as in (1) or (2), or with four requests at a single module. If we start a cycle as in (2), both modules have a request waiting at the end of the cycle. The next cycle cannot begin with all four requests at a single module.

We must use a definition of state that allows us to determine how the access requests that must wait during a cycle are distributed among the modules at the end of the cycle, in addition to telling us how many modules will be busy during the cycle. (1) and (2) above illustrate exactly how this is done. Each state specifies how the p requests are distributed among the modules, without regard to which module is which. It is unimportant in (1) to know which module has three requests and which has two, since the modules and processors are identical.

With this general definition of state, computing the state transition probabilities can be divided into two parts. The first is simple: if $r \leq p$ requests are serviced during a memory cycle, the r processors that generate new requests for the next memory cycle choose any particular set of modules with probability $\left(\frac{1}{m}\right)^r$. They may all choose to send their new requests to the same module, or each may select a different module from the others, or some may choose the same module while others select different modules. The end result will be some distribution of p requests among the m modules at the beginning of the next cycle, with each possible distribution represented by a different state.

The second part of computing state transition probabilities is to count the number of ways in which processors may choose to make new requests that will result in the same distribution of requests at the beginning of the next cycle (the same next state). Consider the following example with 4 processors and 2 memory modules, as above. Suppose the model begins one cycle with two requests at each module. At the end of the cycle, each module has one request remaining. Two processors will make new requests before the start of the next cycle. If they select the same module, the next state has three requests at one module and one request at the other. There are two ways that this can happen, since there are two modules that they can select and each of these modules has one request remaining from the previous cycle.

To put the two parts together, we compute the single step state transition probability for the transition from state A to state B by multiplying the probability of choosing any set of modules following the cycle beginning in state A by the number of ways we can choose a set of modules that takes us to state B. For the example, the probability of making the transition from the state with two requests at each module to the state with three requests at one module and one at the other is $\left(\frac{1}{2}\right)^2 2 = 1/2$.

Generally, the process of computing state transition probabilities is more difficult that this example would indicate, even though the process just described always works. The difficulty will always come in the second part - counting the number of ways of choosing where new requests go that take the model to the same next state. For this reason, we recommend that you follow the above procedure to compute **all** single step transition probabilities for each state. Because the single step transition

probabilities for a given state (including a transition back to the same state) must sum to 1, this will provide a useful check on whether or not you counted all the possibilities.

We illustrate the complete procedure with several examples. Let $C(n, r)$ be the number of combinations of n objects taken r at a time, and $P(n, r)$ be the number of permutations of n objects taken r at a time. In the first two examples, we find an expression for the average memory module concurrency as a function of the number of memory modules for a fixed number (2 or 3) of processors.

Example:
2 processors, m≥2 modules

State	
1	2 requests at the same module
2	1 request at each of two modules

First, find the single step transition probabilities. At the risk of overkill for this particularly simple case, we explicitly write each probability as the product of the probability that the new requests go to a particular set of modules ($\frac{1}{m}$ if the cycle started in state 1 and $\frac{1}{m^2}$ if the cycle started in state 2) and the number of such sets that result in the specified next state.

$$p_{1,1} = \frac{1}{m} 1 = \frac{1}{m}$$

$$p_{1,2} = \frac{1}{m} (m-1) = \frac{m-1}{m}$$

$$p_{2,1} = \frac{1}{m^2} C(m, 1) = \frac{1}{m}$$

$$p_{2,2} = \frac{1}{m^2} C(m, 2) = \frac{m-1}{m}$$

This may actually seem to be double overkill. Since we know that the P matrix is singular, we only need one of the Chapman-Kolmogorov equations, and hence only two of the single step transition probabilities - either $p_{1,1}$ and $p_{2,1}$ or $p_{1,2}$ and $p_{2,2}$. The other independent equation is the normalization equation. However, as mentioned above, if you compute all of the single-step state transition probabilities, you can add the probabilities for all transitions from each state as a check on having gotten them correct.
Continuing with the example, choose the first Chapman-Kolmogorov equation. Then

$$\boldsymbol{\pi}_1 = \boldsymbol{\pi}_1 p_{1,1} + \boldsymbol{\pi}_2 p_{2,1} = \boldsymbol{\pi}_1 \frac{1}{m} + \boldsymbol{\pi}_2 \frac{1}{m}$$

$$\boldsymbol{\pi}_2 = (m-1)\boldsymbol{\pi}_1$$

$$(\boldsymbol{\pi}_1 + \boldsymbol{\pi}_2 = 1 = m\boldsymbol{\pi}_1) \Rightarrow \left(\boldsymbol{\pi}_1 = \frac{1}{m}\right) \wedge \left(\boldsymbol{\pi}_2 = \frac{m-1}{m}\right)$$

The average memory module concurrency or parallelism is

$$B = 1 \times \frac{1}{m} + 2\frac{m-1}{m} = \frac{2m-1}{m}$$

Example:
3 processors, m≥3 modules

State	
1	3 requests at one module
2	2 requests at one module, 1 request at another module
3	1 request at each of three modules

$$p_{1,1} = \frac{1}{m} 1 = \frac{1}{m}$$

$$p_{1,2} = \frac{1}{m} C(m-1,1) = \frac{1}{m}(m-1)$$

$$p_{1,3} = 0$$

$$p_{2,1} = \left(\frac{1}{m}\right)^2 1 = \left(\frac{1}{m}\right)^2$$

$$p_{2,2} = \left(\frac{1}{m}\right)^2 (C(m-1,1) + C(m-1,1)C(2,1)) = \left(\frac{1}{m}\right)^2 3(m-1)$$

$$p_{2,3} = \left(\frac{1}{m}\right)^2 C(m-1,2)P(2,2) = \left(\frac{1}{m}\right)^2 (m-1)(m-2)$$

$$p_{3,1} = \left(\frac{1}{m}\right)^3 C(m,1) = \left(\frac{1}{m}\right)^3 m$$

$$p_{3,2} = \left(\frac{1}{m}\right)^3 C(m,2)C(3,2)P(2,2) = \left(\frac{1}{m}\right)^3 3m\,(m-1)$$

$$p_{3,3} = \left(\frac{1}{m}\right)^3 C(m,3)P(3,3) = \left(\frac{1}{m}\right)^3 m\,(m-1)\,(m-2)$$

$p_{2,2}$ merits an explanation. Two memory modules are busy during a cycle that starts in state 2. At the end of the cycle, the two processors that had their memory access requests serviced will make new requests. The first term inside the square brackets corresponds to the two new requests going to the same memory module, one of the $m-1$ modules without a request at the end of the cycle. There are $C(m-1,1)$ ways to select this module. The second term corresponds to one of the new requests going to the module that still has a request pending, and the other new request going to one of the other $m-1$ modules. There are $C(m-1,1)$ ways to choose the module without a pending request, and there are two ways to order the two new requests so that one goes to the module that already has a request and the other goes to the module that doesn't.
We will make use of the Chapman-Kolmogorov equations for $\boldsymbol{\pi}_1$ and $\boldsymbol{\pi}_3$, plus the normalization equation.

$$\boldsymbol{\pi}_1 = \boldsymbol{\pi}_1 \frac{1}{m} + \boldsymbol{\pi}_2 \frac{1}{m^2} + \boldsymbol{\pi}_3 \frac{1}{m^2}$$

$$\boldsymbol{\pi}_3 = \boldsymbol{\pi}_2 \frac{(m-1)\,(m-2)}{m^2} + \boldsymbol{\pi}_3 \frac{(m-1)\,(m-2)}{m^2}$$

$$1 = \boldsymbol{\pi}_1 + \boldsymbol{\pi}_2 + \boldsymbol{\pi}_3$$

Solving these three equations gives

$$\boldsymbol{\pi}_1 = \frac{1}{m^2 - m + 1}$$

$$\boldsymbol{\pi}_2 = \frac{(3m-2)\,(m-1)}{m\,(m^2 - m + 1)}$$

$$\boldsymbol{\pi}_3 = \frac{(m-1)^2\,(m-2)}{m\,(m^2 - m + 1)}$$

$$B = 1 \times \frac{1}{m^2 - m + 1} + 2\frac{(3m-2)\,(m-1)}{m\,(m^2 - m + 1)} + 3\frac{(m-1)^2\,(m-2)}{m\,(m^2 - m + 1)} = \frac{3m^3 - 6m^2 + 6m - 2}{m\,(m^2 - m + 1)}$$

Example:
4 processors, 4 modules

State	
1	all four requests are at one module
2a	three requests are at one module and one request is at another module
2b	two requests are at one module and two requests are at another module
3	two requests are at one module and one request is at each of two other modules
4	one request is at each of four modules

We've chosen the state names to make explicit the number of memory modules busy in each state. The state transition probabilities are

$$p_{1,1} = \frac{1}{4}$$

$$p_{1,2a} = \frac{1}{4}C(3,1) = \frac{3}{4}$$

$$p_{1,2b} = p_{1,3} = p_{1,4} = 0$$

$$p_{2a,1} = \left(\frac{1}{4}\right)^2$$

$$p_{2a,2a} = \left(\frac{1}{4}\right)^2 C(3,1)P(2,2) = \frac{6}{16}$$

$$p_{2a,2b} = \left(\frac{1}{4}\right)^2 C(3,1) = \frac{3}{16}$$

$$p_{2a,3} = \left(\frac{1}{4}\right)^2 C(3,2)P(2,2) = \frac{6}{16}$$

$$p_{2a,4} = 0$$

$$p_{2b,1} = 0$$

$$p_{2b,2a} = \left(\frac{1}{4}\right)^2 C(2,1) = \frac{2}{16}$$

$$p_{2b,2b} = \left(\frac{1}{4}\right)^2 P(2,2) = \frac{2}{16}$$

$$p_{2b,3} = \left(\frac{1}{4}\right)^2 C(2,1) + \left(\frac{1}{4}\right)^2 C(2,1)C(2,1)P(2,2) = \frac{10}{16}$$

$$p_{2b,4} = \left(\frac{1}{4}\right)^2 P(2,2) = \frac{2}{16}$$

$$p_{3,1} = \left(\frac{1}{4}\right)^3 = \frac{1}{64}$$

$$p_{3,2a} = \left(\frac{1}{4}\right)^3 C(3,1) + \left(\frac{1}{4}\right)^3 C(3,1)C(3,2) = \frac{12}{64}$$

$$p_{3,2b} = \left(\frac{1}{4}\right)^3 C(3,1)C(3,2) = \frac{9}{64}$$

$$p_{3,3} = \left(\frac{1}{4}\right)^3 C(3,2)P(3,3) + \left(\frac{1}{4}\right)^3 C(3,2)C(3,2)P(2,2) = \frac{36}{64}$$

$$p_{3,4} = \left(\frac{1}{4}\right)^3 P(3,3) = \frac{6}{64}$$

$$p_{4,1} = \left(\frac{1}{4}\right)^4 C(4,1) = \frac{4}{256}$$

$$p_{4,2a} = \left(\frac{1}{4}\right)^4 C(4,2)C(4,3)P(2,2) = \frac{48}{256}$$

$$p_{4,2b} = \left(\frac{1}{4}\right)^4 C(4,2)C(4,2) = \frac{36}{256}$$

$$p_{4,3} = \left(\frac{1}{4}\right)^4 C(4,3)C(3,1)C(4,2)P(2,2) = \frac{144}{256}$$

$$p_{4,4} = \left(\frac{1}{4}\right)^4 P(4,4) = \frac{24}{256}$$

Combining all of these probabilities into the single-step transition probability matrix:

$$P = \begin{matrix} 1/4 & 3/4 & 0 & 0 & 0 \\ 1/16 & 6/16 & 3/16 & 6/16 & 0 \\ 0 & 2/16 & 2/16 & 10/16 & 2/16 \\ 1/64 & 12/64 & 9/64 & 36/64 & 6/64 \\ 4/256 & 48/256 & 36/256 & 144/256 & 24/256 \end{matrix}$$

Solve $\boldsymbol{\pi} P = \boldsymbol{\pi}$ plus the normalization equation.
The solution is

$$
\begin{aligned}
\boldsymbol{\pi}_1 & \\
\boldsymbol{\pi}_{2\mathrm{a}} & \\
\boldsymbol{\pi}_{2\mathrm{b}} & = \\
\boldsymbol{\pi}_3 & \\
\boldsymbol{\pi}_4 &
\end{aligned}
\begin{aligned}
0.0323 \\
0.2419 \\
0.1452 \\
0.5081 \\
0.0726
\end{aligned}
$$

The average memory module concurrency is

$$B = \boldsymbol{\pi}_1 1 + (\boldsymbol{\pi}_{2\mathrm{a}} + \boldsymbol{\pi}_{2\mathrm{b}})2 + \boldsymbol{\pi}_3 3 + \boldsymbol{\pi}_4 4 = 2.2610$$

The three examples allow us to compare the average memory module concurrency with four modules for two, three, and four processors:

p	B
2	1.750
3	2.269
4	2.261

In going from 2 to 3 processors, we see about a 30% increase in memory module concurrency. When we go from 3 processors to 4, the performance improves only about 15.5%.

Example - Two Basic Rules of Probability
This module introduces the multiplication and addition rules used when calculating probabilities.

The Multiplication Rule

If A and B are two events defined on a **sample space**, then:
$\mathrm{P(A\ AND\ B)} = \mathrm{P(B)} \cdot \mathrm{P(A|B)}$.

This rule may also be written as : $\mathrm{P(A|B)} = \frac{\mathrm{P(A\ AND\ B)}}{\mathrm{P(B)}}$

(The probability of A given B equals the probability of A and B divided by the probability of B.)

If A and B are **independent**, then $\mathrm{P(A|B)} = \mathrm{P(A)}$. Then
$\mathrm{P(A\ AND\ B)} = \mathrm{P(A|B)}\ \mathrm{P(B)}$ becomes $\mathrm{P(A\ AND\ B)} = \mathrm{P(A)}\ \mathrm{P(B)}$.

The Addition Rule

If A and B are defined on a sample space, then:
$\mathrm{P(A\ OR\ B)} = \mathrm{P(A)} + \mathrm{P(B)} - \mathrm{P(A\ AND\ B)}$.

If A and B are **mutually exclusive**, then $\mathrm{P(A\ AND\ B)} = 0$. Then
$\mathrm{P(A\ OR\ B)} = \mathrm{P(A)} + \mathrm{P(B)} - \mathrm{P(A\ AND\ B)}$ becomes
$\mathrm{P(A\ OR\ B)} = \mathrm{P(A)} + \mathrm{P(B)}$.

Example:
Klaus is trying to choose where to go on vacation. His two choices are: A = New Zealand and B = Alaska

- Klaus can only afford one vacation. The probability that he chooses A is $\mathrm{P(A)} = 0.6$ and the probability that he chooses B is $\mathrm{P(B)} = 0.35$.
- $\mathrm{P(A\ and\ B)} = 0$ because Klaus can only afford to take one vacation
- Therefore, the probability that he chooses either New Zealand or Alaska is $\mathrm{P(A\ OR\ B)} = \mathrm{P(A)} + \mathrm{P(B)} = 0.6 + 0.35 = 0.95$. Note that the probability that he does not choose to go anywhere on vacation must be 0.05.

Example:

Carlos plays college soccer. He makes a goal 65% of the time he shoots. Carlos is going to attempt two goals in a row in the next game.
A = the event Carlos is successful on his first attempt. $\mathrm{P(A)} = 0.65$. B = the event Carlos is successful on his second attempt. $\mathrm{P(B)} = 0.65$. Carlos tends to shoot in streaks. The probability that he makes the second goal **GIVEN** that he made the first goal is 0.90.

Exercise:

Problem: What is the probability that he makes both goals?

Solution:

The problem is asking you to find $\mathrm{P(A\ AND\ B)} = \mathrm{P(B\ AND\ A)}$. Since $\mathrm{P(B|A)} = 0.90$:

Equation:

$$\mathrm{P(B\ AND\ A)} = \mathrm{P(B|A)\ P(A)}\ =\ 0.90*0.65 = 0.585$$

Carlos makes the first and second goals with probability 0.585.

Exercise:

Problem:

What is the probability that Carlos makes either the first goal or the second goal?

Solution:

The problem is asking you to find $\mathrm{P(A\ OR\ B)}$.

Equation:

$$\mathrm{P(A\ OR\ B)} = \mathrm{P(A)} + \mathrm{P(B)} - \mathrm{P(A\ AND\ B)} = 0.65 + 0.65 - 0.585 = 0.715$$

Carlos makes either the first goal or the second goal with probability 0.715.

Exercise:

Problem: Are A and B independent?

Solution:

No, they are not, because $\mathrm{P(B\ AND\ A)}\ =\ 0.585$.

Equation:

$$\mathrm{P(B)} \cdot \mathrm{P(A)} = (0.65) \cdot (0.65) = 0.423$$

Equation:

$$0.423 \neq 0.585 = \mathrm{P(B\ AND\ A)}$$

So, $\mathrm{P(B\ AND\ A)}$ is **not** equal to $\mathrm{P(B)} \cdot \mathrm{P(A)}$.

Exercise:

Problem: Are A and B mutually exclusive?

Solution:

No, they are not because $\mathrm{P(A\ and\ B)}$ = 0.585.

To be mutually exclusive, $\mathrm{P(A\ AND\ B)}$ must equal 0.

Example:
A community swim team has **150** members. **Seventy-five** of the members are advanced swimmers. **Forty-seven** of the members are intermediate swimmers. The remainder are novice swimmers. **Forty** of the advanced swimmers practice 4 times a week. **Thirty** of the intermediate swimmers practice 4 times a week. **Ten** of the novice swimmers practice 4 times a week. Suppose one member of the swim team is randomly chosen. Answer the questions (Verify the answers):

Exercise:

Problem: What is the probability that the member is a novice swimmer?

Solution:

$\frac{28}{150}$

Exercise:

Problem: What is the probability that the member practices 4 times a week?

Solution:

$\frac{80}{150}$

Exercise:

Problem:

What is the probability that the member is an advanced swimmer and practices 4 times a week?

Solution:

$\frac{40}{150}$

Exercise:

Problem:

What is the probability that a member is an advanced swimmer and an intermediate swimmer? Are being an advanced swimmer and an intermediate swimmer mutually exclusive? Why or why not?

Solution:

$\text{P(advanced AND intermediate)} = 0$, so these are mutually exclusive events. A swimmer cannot be an advanced swimmer and an intermediate swimmer at the same time.

Exercise:

Problem:

Are being a novice swimmer and practicing 4 times a week independent events? Why or why not?

Solution:

No, these are not independent events.

Equation:

$$\text{P(novice AND practices 4 times per week)} = 0.0667$$

Equation:

$$\text{P(novice)} \cdot \text{P(practices 4 times per week)} = 0.0996$$

Equation:

$$0.0667 \neq 0.0996$$

Example:
Studies show that, if she lives to be 90, about 1 woman in 7 (approximately 14.3%) will develop breast cancer. Suppose that of those women who develop breast cancer, a test is negative 2% of the time. Also suppose that in the general population of women, the test for breast cancer is negative about 85% of the time. Let B = woman develops breast cancer and let N = tests negative. Suppose one woman is selected at random.

Exercise:

Problem:

What is the probability that the woman develops breast cancer? What is the probability that woman tests negative?

Solution:

$\mathrm{P(B)} = 0.143$; $\mathrm{P(N)} = 0.85$

Exercise:

Problem:

Given that the woman has breast cancer, what is the probability that she tests negative?

Solution:

$\mathrm{P(N|B)} = 0.02$

Exercise:

Problem:

What is the probability that the woman has breast cancer AND tests negative?

Solution:

$\mathrm{P(B\ AND\ N)} = \mathrm{P(B)} \cdot \mathrm{P(N|B)} = (0.143)\ \cdot\ (0.02) = 0.0029$

Exercise:

Problem:

What is the probability that the woman has breast cancer or tests negative?

Solution:

$\mathrm{P(B\ OR\ N)} = \mathrm{P(B)} + \mathrm{P(N)} - \mathrm{P(B\ AND\ N)} = 0.143 + 0.85 - 0.0029 = 0.9901$

Exercise:

Problem: Are having breast cancer and testing negative independent events?

Solution:

No. $\mathrm{P(N)} = 0.85$; $\mathrm{P(N|B)} = 0.02$. So, $\mathrm{P(N|B)}$ does not equal $\mathrm{P(N)}$

Exercise:

Problem: Are having breast cancer and testing negative mutually exclusive?

Solution:

No. $\mathrm{P(B\ AND\ N)} = 0.0029$. For B and N to be mutually exclusive, $\mathrm{P(B\ AND\ N)}$ must be 0.

Glossary

Independent Events
: The occurrence of one event has no effect on the probability of the occurrence of any other event. Events A and B are independent if one of the following is true: (1). $P(A|B) = P(A)$; (2) $P(B|A) = P(B)$; (3) $P(A \text{ and } B) = P(A)P(B)$.

Mutually Exclusive
: An observation cannot fall into more than one class (category). Being in more than one category prevents being in a mutually exclusive category.

Sample Space
: The set of all possible outcomes of an experiment.

Open Education Cup: Authoring in Connexions

Connexions provides a number of help documents designed to assist authors looking to get started creating and editing content. These documents range from basic introductions to the Connexions platform to advanced CNXML tutorials for experienced authors wishing to take their materials to the next level.

Here are a few resources to help you get started:

- Connexions Help Page - http://cnx.org/help/
- Introduction to Connexions - http://cnx.org/content/m14206/latest/
- New Author Guide - http://cnx.org/help/authorguide
- Editing Modules - http://cnx.org/content/m10887/latest/
- Basic CNXML - http://cnx.org/content/m14394/latest/
- Advanced CNXML - http://cnx.org/content/m14395/latest/
- Tables in Connexions - http://cnx.org/content/m14396/latest/
- Multimedia in Connexions - http://cnx.org/content/m12660/latest/

Note: Due to space limitations, not all of the help documents have been included in this collection. You can view any of the documents listed at the URLs indicated above.

If you have any questions, please contact Jonathan Emmons, Connexions' Community Development Specialist, at cnx@cnx.org.

Basic CNXML
This document introduces simple CNXML tags that are easy to use in Edit-in-Place.

Note: This module also contains information derived from The Advanced CNXML by Ricardo Radaelli-Sanchez.

Starting with CNXML

To create the bare bones of content in Connexions, the author interface provides a variety of creation tools: the Document Importer, Edit-In-Place, and even a full-source editor. However, a basic knowledge of our markup language can help make small edits into tremendous enhancements to your material!

Connexions uses the **Connexions Markup Language** (**CNXML**) as its primary language for marking up and storing documents. CNXML is lightweight XML for marking up educational content. Unlike the well-known HTML, the goal of CNXML is to convey the **content** of the material and not a particular presentation. For example, say you have the following sentence: I like cupcakes very much. However, you feel that you enthusiasm for cupcakes has not been fully expressed. In HTML, you would use bold, underline, italic, etc.; in CNXML, you would use the emphasis **tag**.

Inline Tags

Inline tags, such as emphasis, are used to embed content and functionality inside the structural tags, such as paragraphs. Some of the more commonly used tags are discussed below.

Emphasis

As mentioned previously, the `emphasis` tag is used to accent certain text. Note that this refers to **semantic** emphasis and not a typeface. Different **stylesheets** can render emphasis with different typefaces.

Example:

```
<para id='intro'>
  Gardenias are my absolute
<emphasis>favorite</emphasis>
  flower. Their petals are soft, and their bloom
has an
  absolutely <emphasis>heavenly</emphasis> scent.
</para>
```

The above markup will display as:
Gardenias are my absolute **favorite** flower. Their petals are soft, and their bloom has an absolutely **heavenly** scent.

Term

The `term` tag is used to mark words or phrases which are being defined. However, its use is confined to either a para or definition tag. The `term` tag has one optional **attribute**: `URL` - a URL specifying the source or definition of the term.

Example:

```
<para id='gardenia'>

  <term
url="http://en.wikipedia.org/wiki/Gardenia">Garden
```

```
ias
  </term> can be tricky to maintain.  The soil
around the
  roots of a <term>gardenia</term> must remain
moist always,
  but too much water can damage the plant.  Also,
  <term>gardenias</term> enjoy the sun, but if the
  <emphasis>foliage</emphasis> gets wet to bring
the gardenia
  into the shade.
</para>
```

The above markup will display as:
Gardenias can be tricky to maintain. The soil around the roots of a **gardenia** must remain moist always, but too much water can damage the plant. Also, **gardenias** enjoy the sun, but if the **foliage** gets wet to bring the gardenia into the shade.

Note

The `note` tag creates an "out of line" note to the reader. The `type` of note is specified by an optional `type` attribute. If a `type` is not specified, the default is `Note`. The `type` attribute can contain any of the following values:

- note
- aside
- warning
- tip
- important

Example:

```
<para id='pollen'>
  Receiving flowers is, on the whole, a wonderful
thing.
  However, sometimes pollen from the flowers can
cause
  problems.  In particular, the clean up of a bit
of
  pollen can be tricky.  <note type="Important">Do
not
  use water when cleaning up pollen!  This can
lead to
  counter-top and clothing stains!</note>  Your
best bet
  is to use a dry method of cleaning with a paper-
towel.
</para>
```

The above markup will display as:
Receiving flowers is, on the whole, a wonderful thing. However, sometimes pollen from the flowers can cause problems. In particular, the clean up of a bit of pollen can be tricky.

Note: Do not use water when cleaning up pollen! This can lead to counter-top and clothing stains!

Your best bet is to use a dry method of cleaning, with a paper-towel, for example.

Link

The `link` tag is the tag in CNXML used for linking to other Connexions modules or collections as well as external links.

- **strength**The Strength attribute can contain the value 1, 2, or 3 specifying the relevance of the link.
- **window**The Window attribute determines the manner in which the link location will be opened. It can contain the values "Replace" or "New". "Replace" will result in the link location opening in the current window replacing the page with the link. "New" will result in the link location opening in a new browser window.
- **url** The URL attribute can contain the web address of the link you wish to reference.
- **document**The Document attribute is used to reference the ids of other Connexions modules or Collections.
- **target-id**The Target-id attribute is used to reference the ids of specific elements within Connexions modules.
- **resource**
- **version**The Version attribute is used to reference a specific version of a Connexions module or collection.

The `target` and `document` attributes can be used together or alone. If both are used then you will link to a particular tag in another document. If only `document` is used, you will link to another document. If only `target` is used, you will link to a particular tag within the current document.

Cite

The `cite` tag is used to refer to non-electronic materials within a document, and primarily contains the title, the author, and/or a page number of a work.

Example:
Cite Example

```
Finally, a good resource is the <cite>Garden
Lover's
Cookbook -- William M. Rice; Paperback</cite>.
```

The above markup will display as:
Finally, a good resource is the *Garden Lover's Cookbook -- William M. Rice; Paperback*.

Quote

The `quote` tag is used to denote that some text directly quotes another source. The quote tag has a `display` attribute which denotes whether the quote is `inline` or `block`.

Example:
Quote Example

```
<para id='plantquote'> Every plant needs a
different amount of water in order to grow well.
<quote display="inline">"If you water each plant
the same, you will always water too much and too
little."</quote> Also, remember the words of Lou
Erickson: <quote id="quote_example"
display='block'>"Gardening requires lots of water
- most of it in the form of perspiration."</quote>
</para>
```

Every plant needs a different amount of water in order to grow well. ""If you water each plant the same, you will always water too much and too little."" Also, remember the words of Lou Erickson: ""Gardening requires lots of water - most of it in the form of perspiration.""

Foreign

The `foreign` tag is used to denote that a word or phrase foreign to the language of the document is being used.

Example:
Foreign Example
`<para id='plantquote2'> All flowers have a scientific name, often derived from Latin. <foreign>Gardenia augusta</foreign> is the name of a type of gardenia found in Japan. </para>` All flowers have a scientific name, often derived from latin. Gardenia augusta is the name of a type of gardenia found in Japan.

Code

The `code` tag is used to insert example computer output/input as either inline text within a paragraph or as a block of text. The `code` tag has a `display` attribute with two possible values:

- `inline` (default) - used to specify code that is inline.
- `block` - used to specify code that should be in a separate block of text.

Example:
Inline Code Example
For now, take a look at what the inline code looks like:

```
<para id='copy'>
  In a unix terminal the command to copy a file is
  <code display='inline'>cp original copy</code>.
</para>
```

In a unix terminal the command to copy a file is `cp original copy`

You will see more about code blocks in Advanced CNXML using Edit-In-Place.

Advanced CNXML
This document explains and elaborates on CNXML tags that you can insert into a Connexions document using Edit-in-Place.

Para

When working in Edit-in-Place, notice that the first item of the "Add Here" drop-down menu is "Paragraph". When you select this item and click `Add Here`, a text box will appear. You can now insert text in the white box, including inline tags. Note the `id="element-143"` in the upper left hand part of the blue box in [link]. `element-143` is the paragraph's unique **ID**, which you can use to refer to the paragraph directly using a link tag. Also, you can find some helpful tips in the upper right-hand corner of the blue box: "Help editing <para>".

A Paragraph Box

When you click "insert and choose paragraph", a box like this should appear.

Example:
Submitted by J. Cameron Cooper

```
<para id='intro'>
  Working on trees or bushes can generate a lot of
limbs and
branches to haul away. If you just carry them,
it'll take
all day. Instead, make a sledge.
</para>
<para id="intro2">
  Find a large, complex branch to make the base of
your
sledge. It should be relatively flat, and broad
and long
enough to make a decent pile; that is, as big or
bigger than
anything else you need to haul away. Green
branches from
hardwoods are best. Place it with the cut end
pointing the
way you want to go. If no single branch is good
enough, two
can be used. Just place their cut ends a couple
feet apart.
</para>
<para id="intro3">
  Then pile on the remaining branches. Most will
naturally
weave together; if not, give 'em a little help.
Once the
pile it a few layers deep, smaller waste, like
weeds or
maybe even leaves can be added to the pile. If it
gets
unstable, another big branch will help.
```

```
</para>
<para id="intro4">
  When you're done, grab the cut end of the bottom
branch,
and maybe the base of one of the other big
branches in the
pile, and drag the thing where you want to go.
You'll be
surprised how much one person can drag!
</para>
<para id="intro5">
  If you have a lot of leaves or similar small
stuff to move,
you can use a similar technique. Get a tarp, toss
the leaves
and weeds and whatnot in the middle, and then drag
the whole
thing away.
</para>
```

which displays as the following:

Working on trees or bushes can generate a lot of limbs and branches to haul away. If you just carry them, it'll take all day. Instead, make a sledge.

Find a large, complex branch to make the base of your sledge. It should be relatively flat, and broad and long enough to make a decent pile; that is, as big or bigger than anything else you need to haul away. Green branches from hardwoods are best. Place it with the cut end pointing the way you want to go. If no single branch is good enough, two can be used. Just place their cut ends a couple feet apart.

Then pile on the remaining branches. Most will naturally weave together; if not, give 'em a little help. Once the pile it a few layers deep, smaller waste, like weeds or maybe even leaves can be added to the pile. If it gets unstable, another big branch will help.

When you're done, grab the cut end of the bottom branch, and maybe the base of one of the other big branches in the pile, and drag the thing where you want to go. You'll be surprised how much one person can drag!

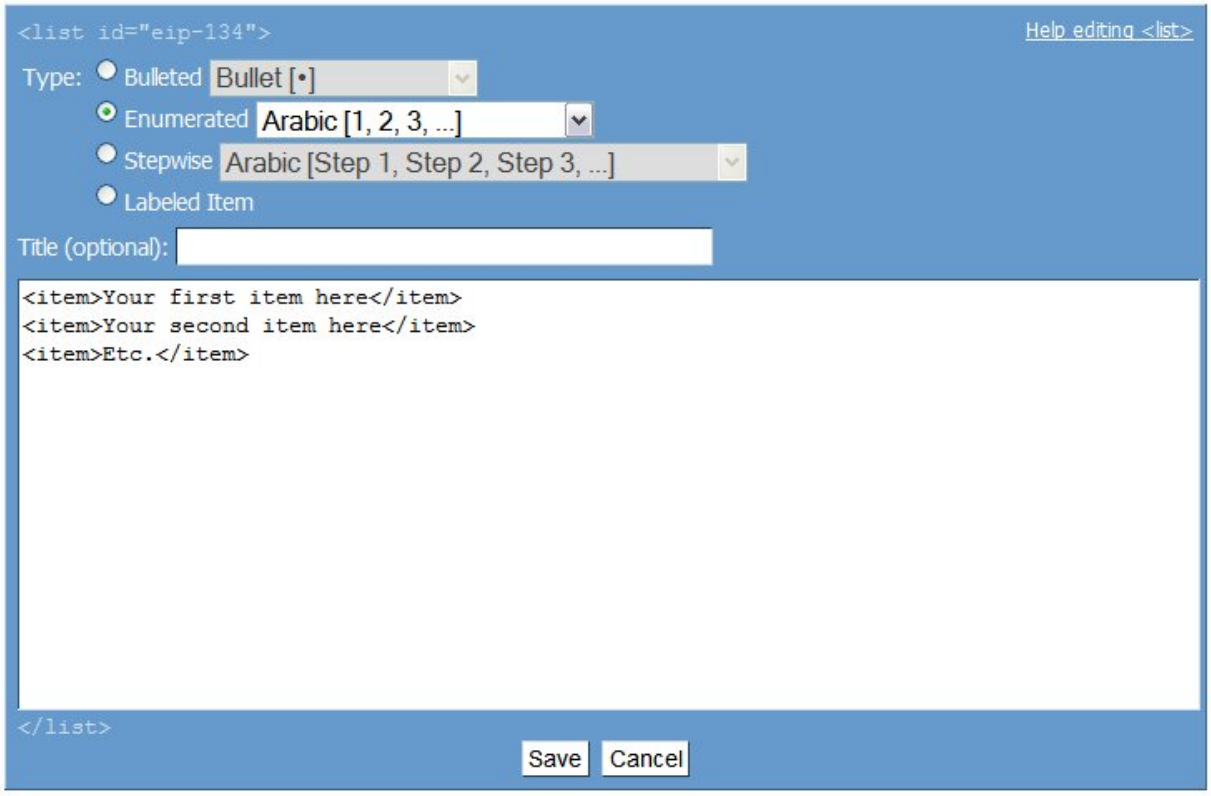
If you have a lot of leaves or similar small stuff to move, you can use a similar technique. Get a tarp, toss the leaves and weeds and whatnot in the middle, and then drag the whole thing away.

List

To insert a new list, select "list" from the "insert" drop-down menu. As with adding a paragraph, adding a list will insert a blue box, with the list's unique ID in the upper left-hand corner and a helpful link in the upper right-hand corner.

Lists Available in Edit-in-Place

Enumerated List



After you add a list, you will see this blue box. You can then select the type of list you wish to use. Here an enumerated list has been selected

Bulleted List

Here a bulleted list has been selected.

Example:
Enumerated List

```
<list id='sledge' list-type='enumerated'>
<title>Making a Sledge</title> <item> Find a
large, complex branch to make the base of your
sledge. It should be relatively flat, and broad
and long enough to make a decent pile; that is, as
big or bigger than anything else you need to haul
away. Green branches from hardwoods are best.
Place it with the cut end pointing the way you
want to go. If no single branch is good enough,
```

```
two can be used. Just place their cut ends a
couple feet apart. </item> <item> Then pile on the
remaining branches. Most will naturally weave
together; if not, give 'em a little help. Once the
pile it a few layers deep, smaller waste, like
weeds or maybe even leaves can be added to the
pile. If it gets unstable, another big branch will
help. </item> <item> When you're done, grab the
cut end of the bottom branch, and maybe the base
of one of the other big branches in the pile, and
drag the thing where you want to go. You'll be
surprised how much one person can drag! </item>
</list>
```
The resulting list will look like:

Making a Sledge

1. Find a large, complex branch to make the base of your sledge. It should be relatively flat, and broad and long enough to make a decent pile; that is, as big or bigger than anything else you need to haul away. Green branches from hardwoods are best. Place it with the cut end pointing the way you want to go. If no single branch is good enough, two can be used. Just place their cut ends a couple feet apart.
2. Then pile on the remaining branches. Most will naturally weave together; if not, give 'em a little help. Once the pile it a few layers deep, smaller waste, like weeds or maybe even leaves can be added to the pile. If it gets unstable, another big branch will help.
3. When you're done, grab the cut end of the bottom branch, and maybe the base of one of the other big branches in the pile, and drag the thing where you want to go. You'll be surprised how much one person can drag!

Example:
Bulleted List

```
<list id="ex-bulleted-list" list-type="bulleted">
  <item>branches</item>
  <item>leaves</item>
  <item>sweat</item>
  <item>lemonade</item>
</list>
```

- branches
- leaves
- sweat
- lemonade

Equation

The `equation` tag is used to set off and number equations in CNXML documents. If you have MathML enabled for your document, you will only be able to place MathML equations within the `equation` tags. Otherwise, to write the actual equations, you can use ASCII or images.

Note: Connexions strongly encourages the use equation with MathML tags when displaying math.

If you look at [link], you will find the equation's unique ID in the upper left-hand corner and a helpful link in the upper right-hand corner.
Adding an Equation

As with lists, you can add an optional `title` at the beginning of each equation.

Example:
Using Images as Equations

```
<equation id="eqn14">
  <media id="img12" display="block" alt="1+2=3"
 <image mime-type='image/gif' src='euler.gif' />
</equation>
```

displays as:
Equation:

1+2=3

Example:

ASCII equations

```
<equation id='eqn15'>
  <title>Simple Arithmetic</title>
    11+27=38
</equation>
```

This equation will display as:

Equation:

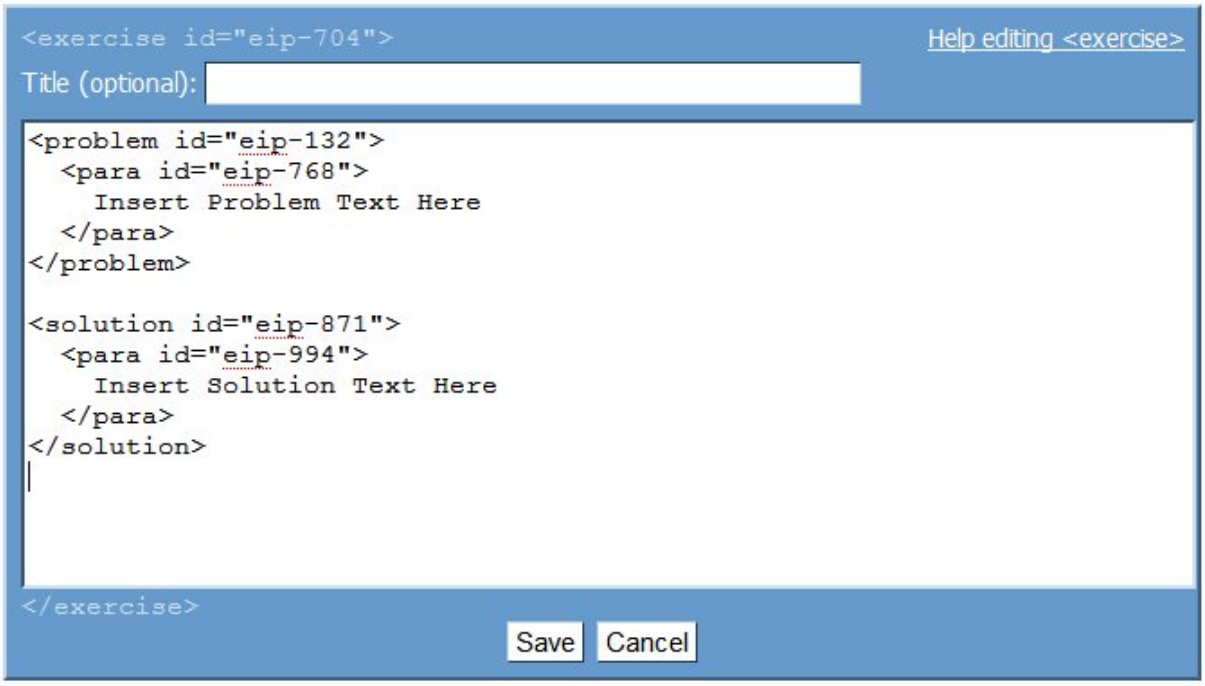
Simple Arithmetic

11+27=38

Exercise

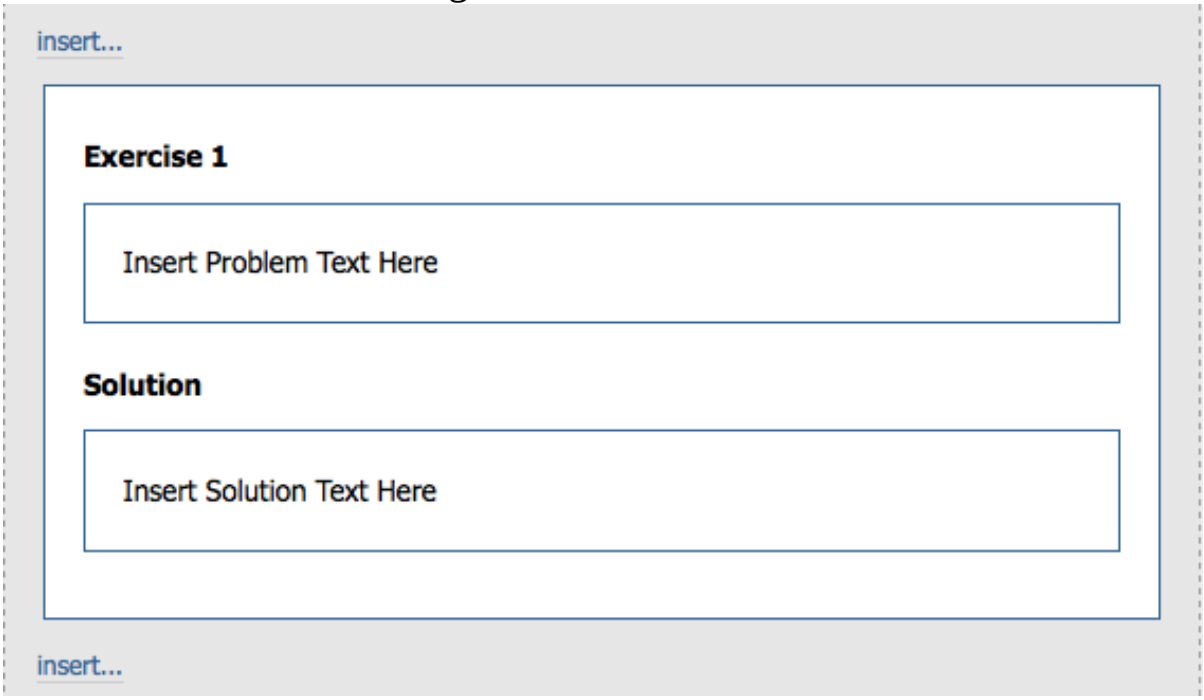
The `exercise` tag allows authors to add practice problems into their documents. When you initially add an exercise, you will see the familiar blue box, with the unique ID and the helpful link in the top corners. However, also notice that new tags have been premade in your text box: `problem` and `solution`.

Adding an Exercise



To continue utilizing edit-in-place to edit your exercise, press the `Save` button (see [link]). You can now add various block tags to your problem and solution, including paragraphs and lists!

A New Exercise after Saving



If you save immediately after creating a new exercise, you can continue to edit the exercise using the familiar edit-in-place interface.

To create more complex exercises, such as multiple-choice, multiple-response, ordered-response, and free-response questions, QML (Questions Markup Language) may used in place of the problem and solution tags. For more information, please see the information about QML.

Example:

```
<exercise id='hyd_test'>
  <problem id="id9">
    <para id='hyd_testp1'>
```

```
      The color of a hydrangea changes with the pH
of the
      soil. What color would the hydrangea be if
the soil
      were highly acidic?  Highly basic?  Neutral?
    </para>
  </problem>
  <solution id="id10">
    <para id='hyd_sol1p1'>
      Highly acidic soil produces blue flowers.
Highly
      basic soil produces pink flowers. Neutral
soil produces
      very pale cream flowers.
    </para>
  </solution>
</exercise>
```

This code will display as:

Exercise:

> **Problem:**
>
> The color of a hydrangea changes with the pH of the soil. What color would the hydrangea be if the soil were highly acidic? Highly basic? Neutral?

> **Solution:**
>
> Highly acidic soil produces blue flowers. Highly basic soil produces pink flowers. Neutral soil produces very pale cream flowers.

Figure

The `figure` tag provides the structure for creating a figure within a document. They can contain either two or more subfigure tags, or a single

media, table, or code tag.

Adding a Figure

Adding a figure will create this familiar blue box, with a helpful link in the upper right corner and the figure's unique ID in quotes in the upper left corner.

The optional first tag of the `figure` tag is title which is used to title a figure.

The `title` tag is followed by any of the tags listed above; however, the most commonly used tag is media, which is used to include any sort of media such as images, video, music, or java applets. For more information on what media you can add to your content, and how to add it, see Adding Multimedia to Your Connexions Content.

The final tag is the optional `caption` which is used to add a small caption to the figure.

Example:
Example of a Figure

```
<figure id='blossom'>
   <title>Momosa Blossom</title>
   <media id="image-example" display="block"
alt="A Momosa Blossom.">
      <image id="flower" mime-type="image/jpeg"
src="alb_jul_flo_1.jpg">
   </media>
   <caption>
      Picture taken by Jenn Drummond (CC
Attribution).
   </caption>
</figure>
```

This code will display as:
Momosa Blossom

Picture taken by Jenn Drummond (CC Attribution).

Code

As seen in Using Basic CNXML in Edit-in-Place, you can add inline code to your document; edit-in-place also allows you in insert a block of code, separate from text.

Adding a Block of Code

Note that

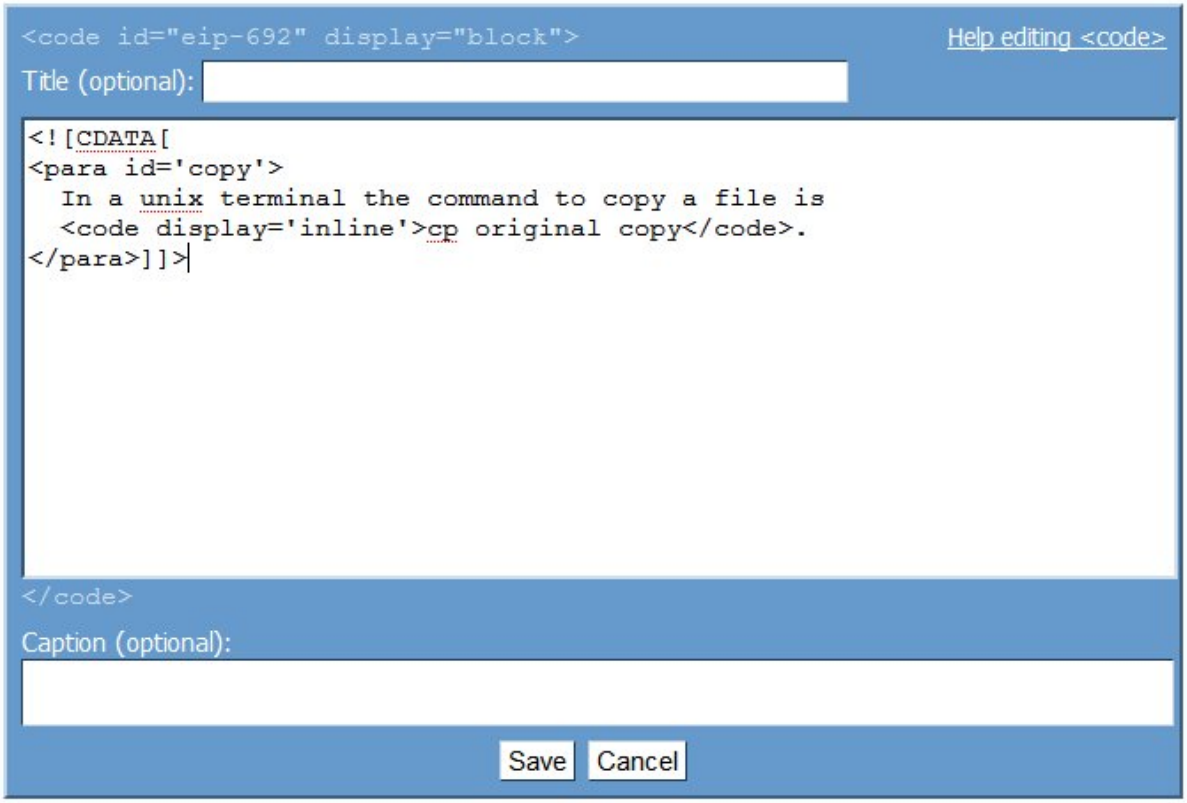
`code`

has a required unique ID **if and only if** the display attribute is **block**.

If you need to use the `>` and `<` symbols in your block of code, you must either use the unicode for these characters (> and <, if you have MathML enabled), or use the CDATA method. To utilize the CDATA method, insert `<![CDATA[` before your code and `]]>` after it, as seen in [link].

Example:
A Block of Code, Using CDATA

Using CDATA in a Code Block



When saved, [link] will display as:

```
<para id='copy'>
  In a unix terminal the command to copy a file is
  <code display='inline'>cp original copy</code>.
</para>
```

Note

As mentioned in Using Basic CNXML in Edit-in-Place, the `note` tag creates an "out of line" note to the reader. You can also insert a note using the drop-down box in Edit-in-Place; however, unless you edit the full source, the type of note will be set to the default.
Adding a Note using Edit-in-Place

As with

`code`

, notes require a unique ID when the display attribute is "block".

Example:

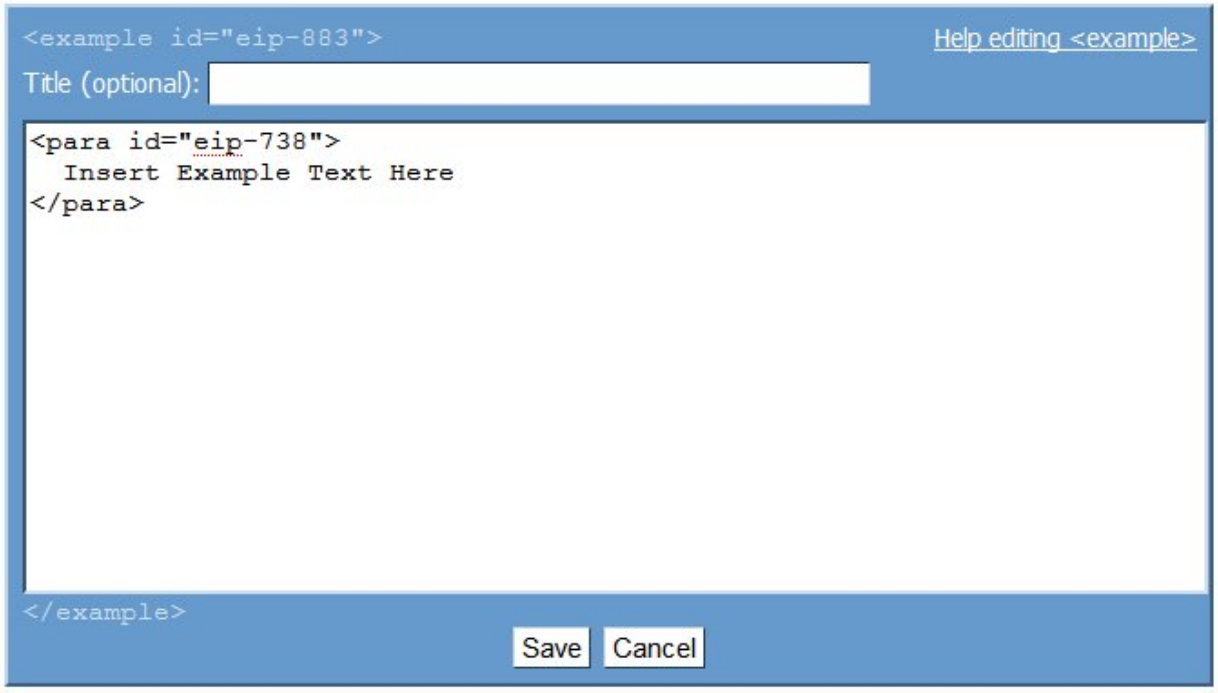
```
<note>
  Gardening requires a lot of intense physical
exertion.
  Please drink plenty of water to avoid
dehydration!
</note>
```

The above markup will display as:

Note: Gardening requires a lot of intense physical exertion. Please drink plenty of water to avoid dehydration!

Example

As is often the case in textbooks, authors will include examples in the middle of a chapter or section. For this reason CNXML provides the example tag that allows an author to include examples in a document.
Adding an Example Using Edit-in-Place



Example:
Here is the code for [link]:

```
<example id="notexamp">
  <code id="codeseg1" display="block">
    <note>
      Gardening requires a lot of intense physical
exertion.
      Please drink plenty of water to avoid
dehydration!
    </note>
  </code>
  <para id="notep2">
    The above markup will display as:
  </para>
  <note>
    Gardening requires a lot of intense physical
exertion.
    Please drink plenty of water to avoid
dehydration!
  </note>
</example>
```

CALS Table

The final element you can add using Edit-in-Place is `table`. To learn more about adding and editting tables using Edit-in-Place, see CALS Table. For a more complete description of the CALS Table consult the CALS Table Spec.